DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

PROYECTO FIN DE CARRERA

Integración de sensores para sistemas ADAS a través de la arquitectura ROS para un vehículo inteligente.

AUTOR: ROBERTO PIQUERAS CEBALLOS
TUTORES: BASAM MUSLEH LANCIS
DANIEL OLMEDA REINO

Leganés, Septiembre de 2014

A todos los que me han ayudado a seguir y no dejaron que me rindiera.

Resumen

En la actualidad la industria automovilística esta centrando sus esfuerzos en el desarrollo de sistemas que ayuden a mejorar la seguridad de las personas. Algunos proyectos persiguen la conducción autónoma del coche, mientras que otros, como es el caso de este proyecto, buscan desarrollar sistemas de ayuda a la conducción.

En el presente trabajo se muestra la implementación por primera vez de disferentes sensores (cámara estéreo, GPS y sensor inercial) mediante el sistema ROS para poder aprovechar los datos recogidos en un proyecto futuro. En primer lugar se toman los datos de los sensores y se obtiene la información que interesa de cada uno. Posteriormente se procesan estos datos y se envian los datos ya procesados para que puedan ser usados por un programa posterior.

Una vez que se tienen los datos, se puede posicionar al coche y los movientos a los que se somete, así como detectar a los peatones que se situen delante del mismo mediante la cámara estéreo.

Indice

Lista de figuras

Lista de abreviaturas

PIB……...Producto Interior Bruto

ADAS…….…….Advanced Driver Assistance Systems

IVIS…….…….In Vehicle Information Systems

ACC…….…….Adaptative Cruise Control

LCA……Lane Change Assistant

LIDAR……..Laser Imaging Detection and Ranging

LDW……Lane Departure Warning system

ABS……Antilock Brake System

TCS…….……Traction Control System

ASR…….…..……..Anti-Slip Regulation

ESP…….…..Electronic Stability Program

EPS……Electric power steering

HUD……..……..Head-Up Display

AHDA……..….Automated Highway Driving Assist

IVVI…….....Intelligent Vehicle Based on Visual Information

ICab…….....Intelligent Car Automobile

DARPA…….…….Defense Advanced Research Projects Agency

CCD…….…….Charge-Coupled Device

CMOS……Complementary Metal-Oxide-Semiconductor

HOG…….…….Histogram of Oriented Gradients

SVM……Support Vector Machine

ROI…….……..Region Of Interest

DGPS……Differential GPS

ROS……Robot Operating System

Open CV……Open Source Computer Vision

HCI……Human Computer Interaction

LISP……..LISt Processing

IDL……..Interactive Data Language

POSIX……Portable Operating System Interface

CAD……Computer-aided design

CATIA…….....Computer-Aided Three dimensional Interactive Application

TCP……Transmission Control Protocol

MRDS……Microsoft Robotics Developer Studio

1. Introducción

Antes de entrar a profundizar en el objetivo del proyecto, es necesario entender porque se invierten tantos recursos y es tan importante la mejora de los sistemas de seguridad en los coches. Para ello se debe conocer la gravedad de los accidentes de tráfico que se producen.

Además, en este capítulo tambien se verán distintos sistemas automáticos, los fundamentos en los que se basan y como han sido desarrollados.

1.1 Motivación para investigar en Sistemas de ayuda a la conducción

Mucho antes de que fueran inventados los automóviles, en las lesiones causadas por el tráfico se veían involucrados carruajes, carros, animales y personas. Las cifras aumentaron exponencialmente con la aparición y constante proliferación de automóviles, autobuses, camiones y otros vehículos de motor. El de un ciclista de la ciudad de Nueva York fue el primer caso registrado de traumatismo en el que participó un vehículo de motor, el 30 de mayo de 1896, y el de un peatón de Londres fue el primer caso registrado de muerte causada por un vehículo de motor, el 17 de agosto de ese mismo año. En 1997, el total acumulado de defunciones causadas por el tráfico se estimó en 25 millones.

Se estima que en 2002 murieron 1,18 millones de personas por causa de colisiones en la vía pública, lo que significa una media de 3.242 fallecimientos diarios. La cifra representa el 2,1% de las defunciones mundiales, lo que convierte a las lesiones causadas por el tráfico en la undécima causa de muerte en el mundo [1].

Además del coste social que suponen estas muertes, los accidentes de tráfico también constituyen un enorme coste económico para los países. Por ejemplo, en 1994, los accidentes de tráfico supusieron para Estados Unidos un coste de 358.022 millones de dólares, un 4,6% de su PIB (Producto Interior Bruto) [2]. En España se realizó una estimación para el año 2002 y se concluyó que el coste de los accidentes de tráfico en ese año había sido de aproximadamente 12.000 millones de euros, lo que supuso un gasto de 400 euros por habitante. Estos costes son la suma de los costes por reparación de los elementos materiales, la pérdida de producción por parte de las personas involucradas en los accidentes, los costes de hospitalizacion y rehabilitación, y los costes administrativos [3].

No obstante, las estadísticas recientes muestran una clara tendencia descendiente en el número de víctimas, especialmente en los países más desarrollados. Se pueden poner como ejemplo los datos representados en la Fig. 1.1 [4]. En esta tabla se puede ver como la tasa de mortalidad sube los primeros años debido al mayor número de vehículos en circulación. En la última década del siglo XX baja considerablemente y se mantiene estable hasta la entrada en el siglo XXI, en el cual comienza la introducción en el mercado de los sistemas modernos de ayuda a la conducción.

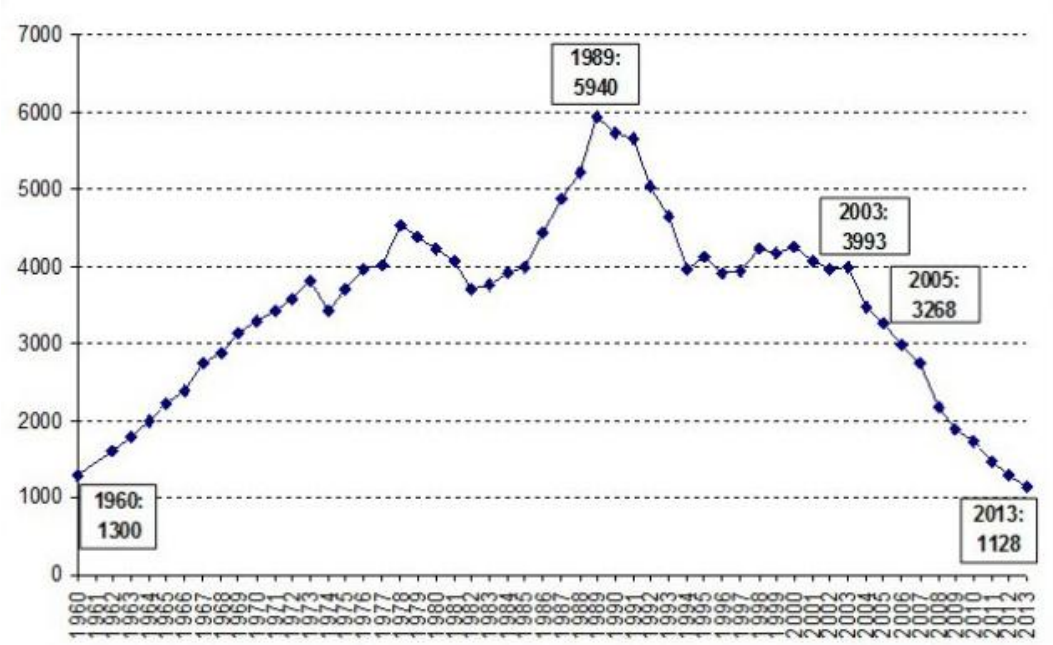


Figura 1.1 Evolución de las victimas por accidente de tráfico en España.

También se puede observar en la Fig. 1.2 como se ha producido un estancamiento en el número de vehículos y de trayectos de largo recorrido realizados y el gran descenso de las víctimas mortales, habiendo disminuido en 2013 a casi un tercio del número de víctimas de 2005.

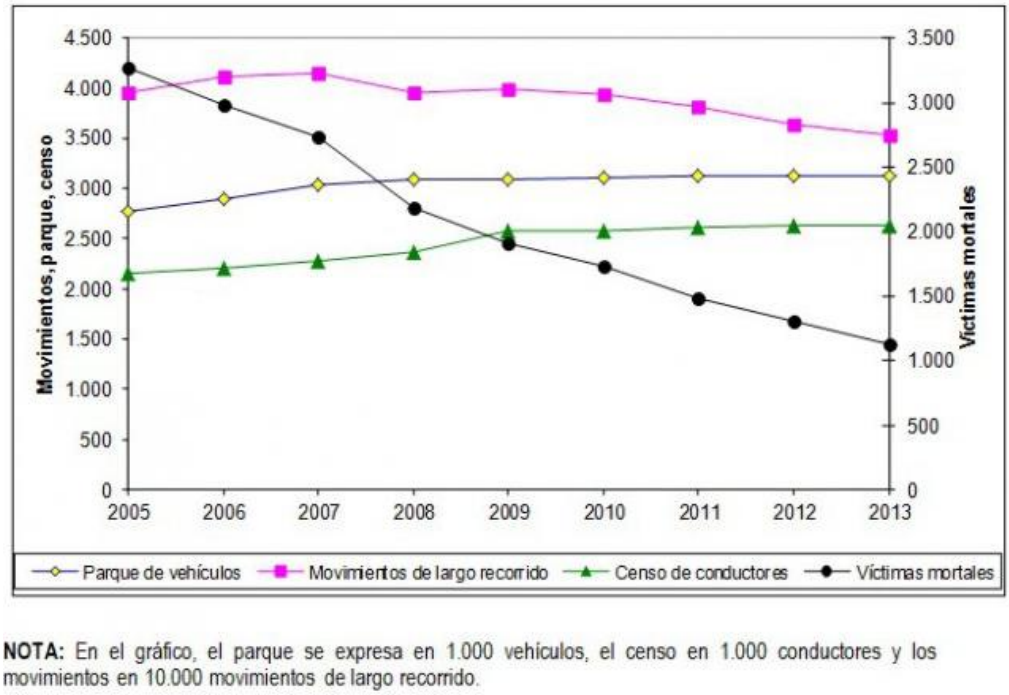


Figura 1.2 Evolución de las víctimas por accidente y el número de vehículos, conductores y desplazamientos largos.

Hay que tener en cuenta que los accidentes de tráfico son, en gran parte, responsabilidad de los conductores. La causa de estos accidentes son los excesos de velocidad, el consumo de alcohol, las distracciones, las infracciones, la imprudencia, etc. Por ello, el colectivo más propenso a sufrir accidentes es el de los jóvenes, ya que estos no suelen ser conscientes de los riesgos que supone el conducir un vehículo. Las campañas de concienciación y la mejora de las infrestructuras han tenido una influencia notable en la reducción del número de accidentes, pero a pesar de esto, el fallo humano sigue existiendo. Por este motivo son tan importantes los sistemas de seguridad que aprovechan la tecnología disponible para prevenir posibles accidentes.En la última década se han extendido los sistemas que, usando conocimientos de multiples ámbitos, pretenden proteger a los ocupantes del vehículo así como a los usuarios fuera del vehículo. Se pueden clasificar estos elementos en elementos de seguridad pasiva y elementos de seguridad activa.

Los elementos de seguridad pasiva son los elementos que reducen al mínimo los daños que se pueden producir cuando el accidente es inevitable [5]. Entre estos, se encuentran el cinturón de seguridad, los Airbags, o el diseño de la carrocería para que absorba la mayor cantidad posible de energía en el momento del golpe por medio de la deformación. En la Fig. 1.3 se puede ver el sistema de Airbags de un automóvil.

Figura 1.3 Elementos de seguridad pasiva.

Los elementos de seguridad activa son todos aquellos destinados a otorgar al vehículo una mayor eficacia y estabilidad, evitando de esta manera que se produzca un accidente. En este grupo se encuentran entre otros, los sistemas de frenado, dirección y suspensión, la iluminación, los neumáticos o los sistemas de control de estabilidad.

Los dispositivos electrónicos de este tipo son tendencia en materia de seguridad por parte de investigadores y de fabricantes de automóviles. El principal objetivo es reducir la posibilidad de un accidente supliendo la falta de atención, concentración y conocimientos de los conductores.

1.2 Sistemas de ayuda a la conducción

Los sistemas cuyo principal objetivo es mejorar la seguridad y/o el confort, ayudando al conductor en su tarea de conducir se denominan ADAS (Advanced Driver Assistance Systems, Sistemas avanzados de asistencia a la conducción) [6]. Algunos ejemplos son los sistemas de ayuda al cambio de carril, de aviso en caso de colisión frontal, monitorización de la fatiga del conductor, etc. En la Fig. 1.4 se ve un esquema del intercambio de información entre los distintos componentes que intervienen en el proceso, pudiendose observar un intercambio de información bidireccional.

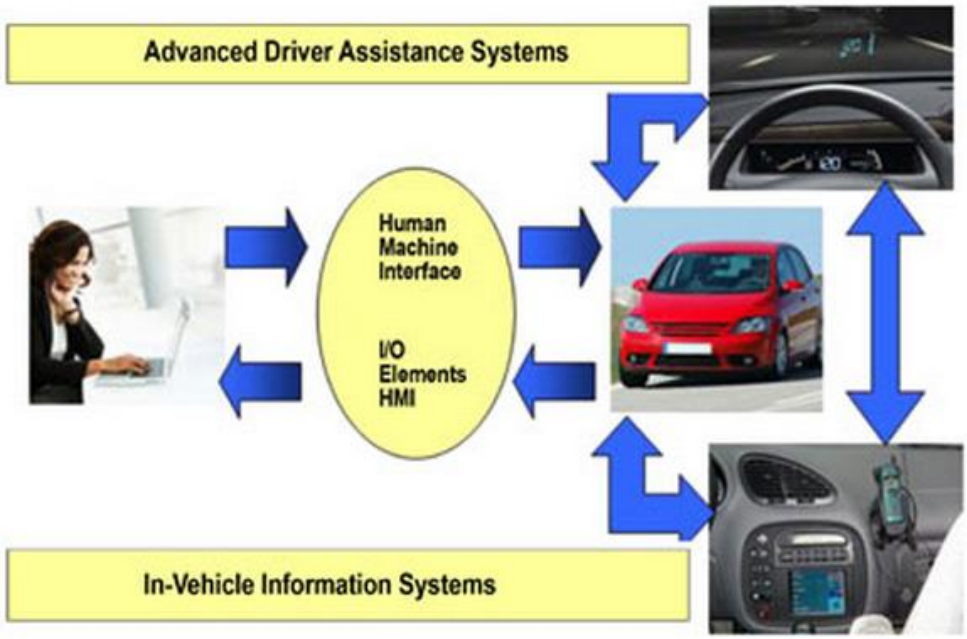


Figura 1.4 Flujo de información de los sistemas de ayuda a la conducción.

Los sistemas que proporcionan información al conductor se denominan IVIS (In-Vehicle Information Systems, Sistemas de información en el vehículo). Estas tareas no tienen por qué estar relacionadas con la tarea principal de conducir, como por ejemplo, servicios telemáticos o de comunicación, navegación, radio, CD, DVD, mp3, dispositivos móviles, etc. Habitualmente estas funciones introducen una segunda tarea que potencialmente puede interferir con la tarea principal de conducción.

La implementación de todas estas funciones y sistemas requieren de la aplicación de multitud de tecnologías. Las principales tecnologías que han adquirido un importante rol dentro del automóvil son:

Tecnología radar: Utiliza ondas electromagnéticas para la detección de distancias y velocidades de objetos. Pueden ser de largo alcance (77 GHz, con una elevada direccionalidad y un alcance de 200 metros) o de corto alcance (24 GHz, con un gran ángulo de detección y alcance de 20 metros). Sus aplicaciones actuales son los sistemas de control de crucero adaptativo (ACC) y los asistentes de cambios de carril (LCA).

Tecnología Lidar: Consiste en un haz pulsado con una única longitud de onda. El haz es estrecho y concentrado, y utiliza el principio de reflexión de las ondas para determinar distancias. Llega a un alcance de 200 metros con una precisión (de mm). Existen dos tipologías diferentes: multibeam (el proceso para la medición de la distancia entre el sensor y el terreno se lleva a cabo mediante la medición del tiempo que tarda un pulso desde que es emitido hasta que es recibido. El emisor funciona emitiendo pulsos de luz) y scanning laser (en este caso el emisor emite un haz láser continuo. Cuando recibe la señal reflejada mide la diferencia de fase entre

la emitida y la reflejada. Conocida ésta, solo hay que resolver el número de longitud de ondas enteras que ha recorrido para conocer la distancia). Las principales aplicaciones estan en la detección de líneas y objetos.

Tecnología Infrarroja: La aplicación de la tecnología infrarroja puede realizarse mediante LED infrarrojo o diodo de láser infrarrojo como emisor (el segundo es más directivo). Las principales aplicaciones son sistemas de visión nocturna (*En los sistemas de visión nocturna en los que se usa una cámara de infrarrojos cercanos, el alcance es de 150 metros, y en los que se usa una cámara de infrarrojos lejanos, el alcance es de 300 metros*) y sistemas de control lateral (LDW).

Tecnología Video: Puede utilizarse una única cámara (mono) o dos cámaras (estéreo). Esta tecnología se usa principalmente para la detección de peatones y líneas.

En la Fig. 1.5 se pueden observar los rangos y posición habitual de los distintos sensores.

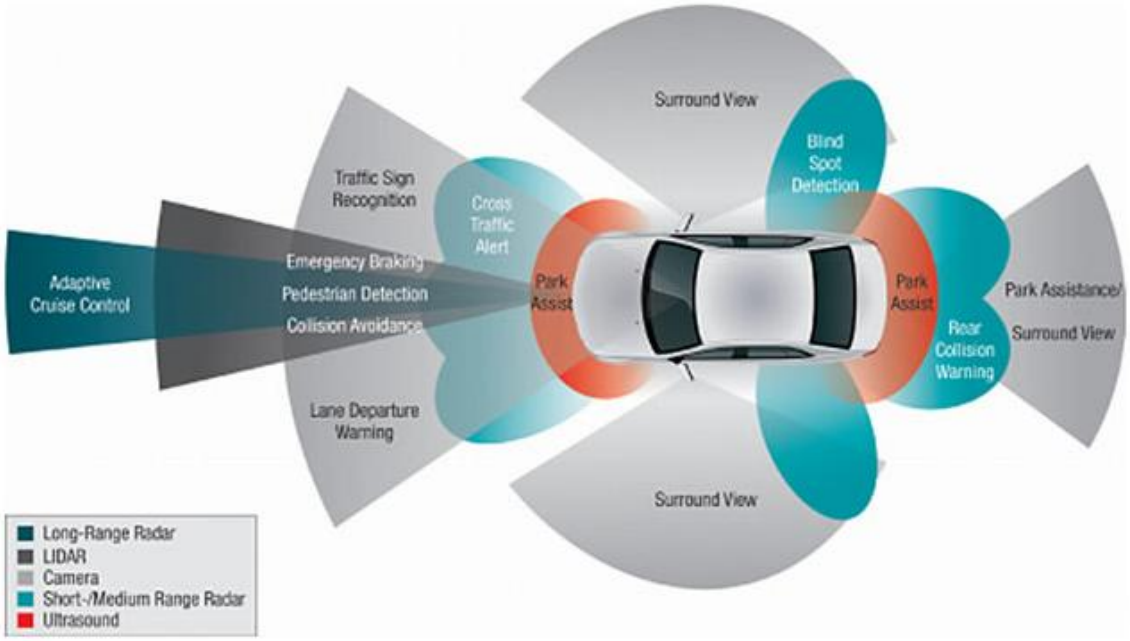


Figura 1.5 Alcance de los distintos sistemas ADAS en función de la tecnología utilizada.

1.2.1 Aplicaciones comerciales

Algunos de los sistemas ADAS más conocidos y que se pueden encontrar actualmente en muchos vehículos comerciales son [7]:

Sistema antibloqueo de frenos (ABS): El ABS evita que las ruedas se bloqueen y patinen ante una frenada fuerte y, por tanto, que se pierda adherencia y estabilidad. Su gran efectividad para evitar accidentes originó que todos los

fabricantes acordaran voluntariamente a partir de 2004 montarlo de serie en sus vehículos destinados al mercado europeo.

Control de tracción (TCS, ASR): Previene la pérdida de adherencia de las ruedas y que éstas patinen cuando el conductor se excede en la aceleración del vehículo o el terreno está muy deslizante. Funciona con los mismos sensores que utiliza el ABS.

Control de estabilidad (ESP, VDC, DSC, ESC, VSC): Detecta si hay riesgo de derrape, e interviene frenando individualmente cada rueda y reduciendo la potencia del motor para restaurar la estabilidad del vehículo. Debe ir equipado en todos los turismos que se hayan fabricado desde noviembre de 2011. El ESP incluye las funciones de ABS y TCS.

Reparto Electrónico de la Frenada (EBV, EBD): A diferencia del ABS, este sistema reparte de forma electrónica la fuerza de frenado entre ejes, y no individualmente a cada rueda. Determina cuánta fuerza hay que aplicar a cada rueda para detener el vehículo en una distancia mínima y sin que se descontrole. Ayuda a que el freno de una rueda no se sobrecargue y evita que el de la otra rueda quede infrautilizado.

Sistema de Control de Velocidad de Crucero Adaptativo (ACC): Al igual que ocurre con el Control de Velocidad de Crucero, regula la velocidad a la que se quiere circular de forma automática. La novedad del ACC es que, con la ayuda de un sistema de radar controla, también de forma automática, la distancia de circulación con respecto al vehículo precedente, frenando el vehículo si es necesario para mantener dicha distancia de seguridad.

Sistema de dirección eléctrica asistida (EPS): A través de sensores el sistema es capaz de registrar el movimiento que el conductor realiza sobre el volante, la velocidad de marcha del propio automóvil y el régimen del motor de combustión. Y en función de estos parámetros una unidad de control eléctrica calcula instantáneamente el par de asistencia necesario en cada momento. Este sistema incrementa el control de la dirección cuando se circula a altas velocidades y facilita las maniobras de aparcamiento a baja velocidad.

Sistema de pre-colisión (PCS): Este sistema reduce los daños y lesiones en una colisión. Reconoce situaciones de accidente inminente y prepara tanto el coche como a los pasajeros para minimizar los daños. Por ejemplo, activa los pretensores de los cinturones de seguridad y ajusta las posiciones de los asientos. Ahora también se ofrece una versión más avanzada que alerta al conductor del peligro de colisión mediante una señal sonora y un aviso en la pantalla de información. Si el

conductor no reacciona, pone en marcha el asistente de frenada de emergencia y activa los frenos automáticamente para reducir la velocidad de impacto.

Detector de ángulo muerto (BLIS, BSM): Utiliza dispositivos de radar montados en las esquinas del parachoques posterior para detectar vehículos que están adelantando por los carriles adyacentes. Existen sistemas que alertan de forma continua de la existencia de vehículos en el ángulo muerto independientemente de las intenciones del conductor mientras que los más efectivos actúan únicamente cuando el conductor activa el intermitente para realizar un cambio de carril.

Aviso de salida de carril (LDW): Mediante sensores infrarrojos situados en la parte inferior del paragolpes delantero o a través de cámaras dinámicas instaladas detrás del parabrisas, el LDW registra y detecta continuamente las marcas viales del carril de circulación. En el caso de un cambio imprevisto de carril, por la ausencia de señalización del intermitente, el sistema alerta al conductor mediante una señal sonora, un testigo luminoso en el cuadro de instrumentos o una vibración en el asiento del conductor o en el volante.

Asistencia de mantenimiento de carril (LKA): Es un sistema inteligente que ayuda al conductor a guiar su coche y garantizar que no se salga del carril de manera involuntaria. Sujeto a las condiciones atmosféricas y al estado de la carretera, la LKA supervisa las líneas blancas de la calzada a través de una cámara estéreo. Este sistema incluye la función de Aviso de salida del carril (LDW).

Detector de peatones con frenada de emergencia: Esta tecnología es capaz de detectar la presencia de un peatón delante del vehículo y si el conductor no responde a tiempo, el vehículo avisa y activa automáticamente los frenos. A través de un radar en la parrilla del coche, una cámara al lado del espejo retrovisor interior y una unidad de control central, el sistema detecta cualquier peatón situado delante del coche al tiempo que calcula la distancia entre ambos. También es capaz de detectar peatones que están a punto de alcanzar la calzada.

Head-Up Display (HUD): Es un dispositivo electrónico que proyecta en el parabrisas del vehículo, a la altura de los ojos, la información más importante del cuadro de instrumentos. Su principal objetivo es evitar que el usuario tenga que desviar la vista de la carretera. Este sistema procede del mundo de la aviación.

Faros de Xenon/Bi-Xenon/Led: Los faros de Xenon son un sistema de iluminación capaz de suministrar el doble de luz que un faro tradicional con sólo dos tercios de la potencia. Además, evita el cansancio del conductor y facilita una conducción más

relajada incluso en trayectos largos. Los faros Bi-Xenón, además, consiguen que la luz de cruce y de carretera tengan un color idéntico, adaptándose de manera óptima a la luz diurna y evitando el deslumbramiento de los vehículos que se aproximan en dirección contraria. Pero hoy en día uno de los avances más significativos en iluminación es la tecnología Led. Consiste en diodos emisores de luz con un consumo energético muy bajo, una larga vida útil y una rápida velocidad de respuesta.

Los sistemas de confort de asistencia al conductor apoyan al conductor durante la conducción. Es el propio conductor quien se encarga de activar muchos de estos sistemas. Algunos de estos sistemas son:

Sistema inteligente de información al conductor (IDIS): Este sistema registra continuamente la actividad del conductor y es capaz de retrasar la entrada de mensajes de texto y llamadas cuando las circunstancias de conducción no son adecuadas y puede haber riesgo de accidente.

Asistente de visión nocturna: Muestra la carretera con la misma intensidad que si estuvieran las luces de carretera conectadas pero sin deslumbrar a los demás conductores. Dos faros infrarrojos de corto alcance situados al lado de la parrilla del radiador emiten una luz invisible al ojo humano pero con un alcance similar al de los faros bixenon. En el interior del coche una cámara sensible a los rayos infrarrojos situada en el parabrisas consigue captar la imagen de la carretera y proyectarla en la pantalla multifunción del vehículo, mostrando posibles obstáculos.

Sistema de reconocimiento de señales de tráfico: Detecta límites variables de velocidad, prohibiciones de adelantamientos y finalización de las mismas. En el futuro será posible detectar otras señales de tráfico. Gracias a este sistema el conductor está continuamente informado de los límites de velocidad de la carretera por la que circula.

Limpiaparabrisas automático: Es uno de los sistemas más frecuentes y conocidos. Mediante un sensor en el parabrisas, este sistema es capaz de detectar presencia de agua, activando entonces de forma automática los limpiaparabrisas y ajustando su frecuencia según la intensidad de la lluvia.

Detector de fatiga en el conductor: Este sistema es capaz de reconocer si el conductor está a punto de dormirse al volante, a través de sensores, y de advertirle antes de que ocurra un accidente. El sistema realiza un seguimiento de varios

aspectos de la cara, incluido el grado de apertura de los ojos. Además analiza los patrones de conducción y las reacciones del conductor y los combina con datos sobre la velocidad de circulación, la hora y el comportamiento del intermitente. Si detecta que hay cansancio en el conductor, le avisa para que se tome un descanso.

Indicador de la presión de los neumáticos (TPM): Un monitor de presión de neumáticos avisa al conductor, mediante un piloto de aviso instalado en el salpicadero, si existe algún neumático con la presión baja.

Control de crucero: Este sistema electrónico permite al conductor fijar una velocidad de circulación sin necesidad de mantener presionado el pedal del acelerador. Si se pulsa de nuevo el botón se recupera automáticamente la velocidad previamente seleccionada y con solo presionar el pedal del freno, este se desactiva.

Sistema de navegación: Es uno de los sistemas de asistencia al conductor más conocidos. Gracias al uso de mapas almacenados y la navegación GPS, el sistema calcula la ruta más rápida al destino elegido.

Asistente al aparcamiento: Este sistema ayuda al conductor durante las maniobras de aparcamiento. Los hay que encuentran el hueco para aparcar, los que facilitan el aparcamiento con cámaras que muestran lo que hay alrededor del coche, e incluso los que mueven el volante y el conductor sólo tiene que accionar los pedales y el cambio.

Asistente de luces largas adaptativas: Ajusta la iluminación del coche en función de la distancia a la que se encuentre el vehículo que circule por el carril contrario. Según se acerque el vehículo, el rango de iluminación irá decreciendo, evitando así que el conductor tenga que cambiar de luces largas a cortas.

Sistema de iluminación adaptativa (AFL): Este sistema combina una luz estática con otra luz giratoria que ilumina la carretera en cruces y curvas cerradas. De esta manera se consigue una mejor iluminación según las necesidades del conductor.

Reconocimiento de voz: En la actualidad se ofrecen sistemas de conectividad en el automóvil que reconocen un número determinado de vocablos que permiten al conductor dar órdenes por voz de llamadas de teléfonos, mensajes de texto, regular el climatizador, etc. Los sistemas de reconocimiento de voz tienen como objetivo evitar al máximo las distracciones del conductor.

Asistente al arranque en pendiente (Hill Holder): Detecta el ángulo de inclinación de la carrocería y evita que el vehículo se vaya hacia atrás en una pendiente durante unos segundos cuando el conductor levanta el pie del freno.

Suspensión Dinámica Adaptativa: Gracias a este sistema, la fuerza de amortiguación de las cuatro ruedas se controla de forma inteligente en función del estado de la carretera y del estilo de conducción. No sólo aumenta el confort de conducción, sino que también mejora la estabilidad y la maniobrabilidad.

A pesar de todos estos sistemas, las compañias siguen investigando para hacer la conducción más segura y confortable. Por ejemplo, Toyota pretende sacar al mercado un nuevo sistema denominado AHDA (Automated Highway Driving Assist). El sistema AHDA combina dos tecnologías de conducción automatizada para hacer posible una conducción más segura y reducir el estrés sobre el conductor: el Control de crucero adaptativo-cooperativo, que se comunica de forma inalámbrica con los vehículos precedentes para mantener la distancia de seguridad, y el Control de trayectoria en carril, que actuando sobre la dirección ayuda a mantener el vehículo en una línea de conducción óptima dentro del carril [8].

1.2.2 Proyectos de investigación

Además de los esfuerzos llevados a cabo por las marcas comerciales para desarrollar sistemas que se puedan aplicar con inmediatez en vehículos de calle, hay numerosos proyectos en curso en todo el mundo que tienen objetivos más ambiciosos, cuyo objetivo es conseguir la conducción totalmente autónoma. A continuación se describen algunos de los más relevantes:

Proyecto de coche autónomo de Google: Google está desarrollando su propio programa de vehículos autónomos [9] y [10]. En agosto de 2012, el equipo de desarrollo anunció que habían sido capaces de circular más de 300.000 millas (unos 480.000 km) de forma autónoma sin sufrir ningún accidente. Google dispone para este proyecto de ocho vehículos: seis Toyota Prius, un Audi TT y un Lexus RX450h como el de la Fig. 1.6. Los vehículos de Google recogen los datos mediante cuatro tipos de sensores: Un lidar giratorio en el techo que genera un mapa tridimensional del entorno a una distancia de 60 m; Una videocámara montada cerca del retrovisor interior que detecta los semáforos y ayuda a los ordenadores a bordo a reconocer el movimiento de obstáculos, como peatones o ciclistas; Un estimador de posición, montado en la rueda trasera izquierda, que mide pequeños movimientos realizados

por el vehículo y ayuda a localizar de manera más precisa su posición en el mapa; Y cuatro radares estándar de automoción, tres en la parte delantera y uno en la trasera, para ayudar a determinar las posiciones de los objetos distantes.

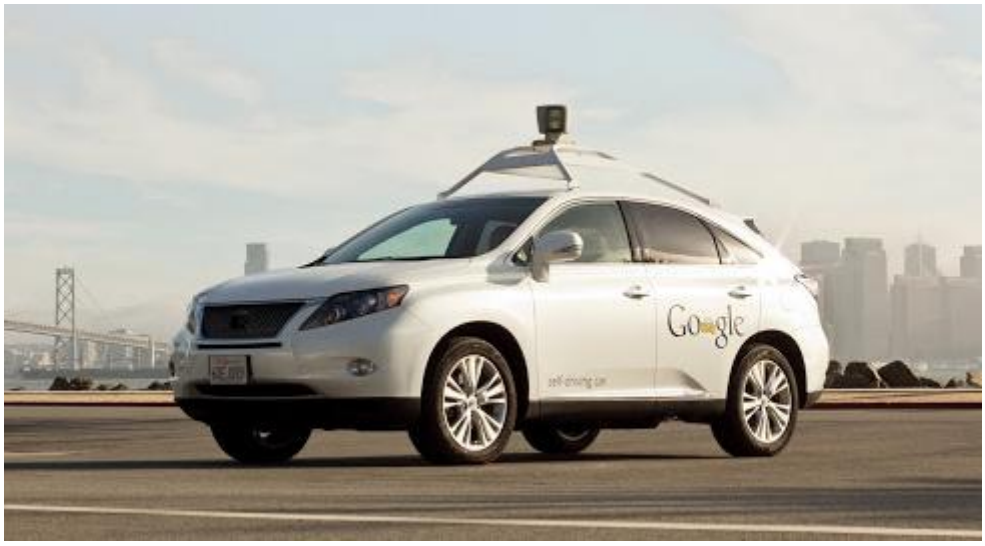


Figura 1.6 Coche Lexus RX450h del proyecto Google

Vehículo de investigación IVVI (Intelligent Vehicle based on Visual Information) de la universidad Carlos III: Utilizando un Nissan Note como base para el desarrollo, se han conseguido instalar distintos sensores en este coche para conseguir realizar las siguientes funciones[11]: Detectar los límites de la carretera y de los carriles, gracias a las cámaras instaladas, y clasificarlos (las imágenes son enviadas al ordenador central, que mediante su software detecta el tipo de carril de que se trata, para poder actuar en consecuencia). Por ejemplo, detecta líneas continuas (donde no se puede adelantar), discontinuas, y carriles de aceleración. Con esta detección ya conseguida, se pretende dar un paso adicional, que es anticipar el peligro ocasionado cuando el conductor se va alejando (inconscientemente) del carril; Detectar y reconocer de forma automática las señales de tráfico, por su forma y color, gracias a sus algoritmos de búsqueda. Este desarrollo se puede aprovechar, entre otras acciones, para la inspección del estado de las señales de tráfico de forma automática, atendiendo al color, forma, posición y tamaño de la misma; Detectar, con las mismas cámaras del coche, a los peatones que puedan suponer un peligro, con el objetivo de poder avisar al conductor (o frenar de forma automática), y de esta forma evitar los accidentes; Reconocer cuando el conductor está comenzando a quedarse dormido con las cámaras que hay dentro del vehículo, las cuales vigilan el rostro de este para detectar si su mirada se desvía, o si la frecuencia de parpadeo no es la habitual (en cuyo caso se alerta al conductor de forma sonora). En la Fig. 1.7 podemos ver tres imágenes de este proyecto. La primera corresponde con las cámaras instaladas, la segunda con una imagen de la detección

de señales y los datos mostrados en consecuencia, y la tercera imagen corresponde con la vista del coche desde fuera.

Figura 1.7 Proyecto IVVI: cámaras (a), datos mostrados (b) y vista exterior (c).

Proyecto Autopia para el desarrollo de vehículos autónomos: En este programa, automóviles Citroën bautizados con nombres clásicos españoles, como Clavileño o Babieca, han protagonizado recorridos notables sin que el conductor intervenga, como el de 100 kilómetros por vías de diversos tipos entre San Lorenzo de El Escorial y Arganda, ambos en Madrid, en 2012 [12]. En Arganda está el Centro de Automática y Robótica (CSIC y Universidad Politécnica de Madrid) en el que se ha desarrollado la mayor parte del trabajo y casi todas las pruebas desde 2002, al amparo de proyectos de investigación españoles, integrados muchas veces en programas europeos para el avance en sistemas inteligentes de transporte. Estos vehículos implementan diferentes sensores: GPS, sistemas de visión, láser y lidar, ultrasonidos, acelerómetros y otros. Pero el objetivo principal no es desarrollar un automóvil concreto, sino aumentar el conocimiento en un tema verdaderamente complejo, el comportamiento del conductor al volante, lo que es generalizable y bastante independiente del vehículo, aunque lógicamente los últimos pasos del proceso, las maniobras como el frenado o la aceleración, dependen de actuadores que sí se adaptan a cada vehículo concreto.

El proyecto Autopía ha avanzado desde su inicio, en el que se abordó la instrumentación con sensores, hasta la automatización de los actuadores del coche (tanto el hardware como

los algoritmos), y está en la fase de conseguir la cooperación entre vehículos, con maniobras como los adelantamientos con tráfico de frente, los cruces sin semáforos y también con semáforos regulables a distancia, la circulación por rotondas o la incorporación a vías rápidas. Pero existen problemas sin resolver, sobre todo si se tiene en cuenta que hay que garantizar la conducción autónoma tanto de día como de noche y en toda clase de condiciones meteorológicas. Todavía hay muchos problemas sin resolver, como por ejemplo que los sensores sean capaces de distinguir las distintas manchas sobre el terreno, entre otras, manchas de aceite o placas de hielo. En la Fig. 1.8 podemos ver imágenes de dos coches que forman parte de este proyecto.

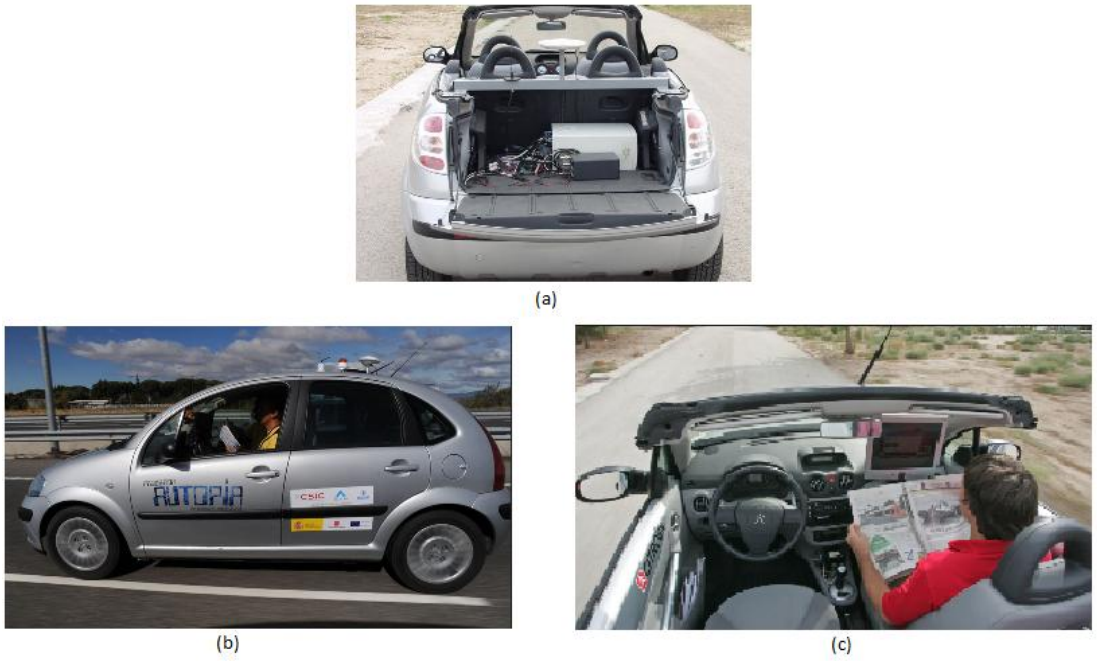


Figura 1.8 Vehículos del programa Autopía: maletero (a), circulación autónoma (b) y (c).

Proyecto iCab: El iCab (Intelligent Car Automobile) surge de un vehículo eléctrico de transporte al cual se le han realizado modificaciones en hardware y software para el control automático de la plataforma (a través de un ordenador) o manual (por medio de un joystick y pedales) [13]. El vehículo eléctrico utilizado como plataforma es un carro de golf de flota de la compañía Ezgo. Este proyecto de I+D+I pertenece a una de las líneas de investigación del Laboratorio de Sistemas Inteligentes (LSI) de la Universidad Carlos III de Madrid e involucra a personas que desarrollan e investigan en busca de la mejora del sistema en su conjunto. El proyecto iCab surge con la primera plataforma (iCab1) y la evolución, para futuras líneas de investigación, es la creación de la segunda plataforma (iCab2). Las plataformas iCab integran un conjunto de elementos que les permiten interactuar y conocer el entorno. Estos

sistemas que se le integran son: un láser, un sistema de visión estereoscópica y un sistema GPS con sensores inerciales. En la Fig. 1.9 se muestra el vehículo utilizado en este proyecto.

Figura 1.9 Vehículo del proyecto iCab

Proyecto Darpa (Grand Challenge): DARPA acrónimo de la expresión en inglés "Defense Advanced Research Projects Agency (Agencia de Proyectos de Investigación Avanzados de Defensa)" es una agencia del Departamento de Defensa de Estados Unidos responsable del desarrollo de nuevas tecnologías para uso militar [14]. Fue creada en 1958 como consecuencia tecnológica de la llamada Guerra Fría. La propuesta de DARPA es imaginar qué capacidades pudiera desear un comandante militar en el futuro, y acelerar estas capacidades de forma concreta a través de demostraciones tecnológicas. Esto no sólo proporciona opciones al comandante, sino que también cambia la mentalidad acerca de lo que es tecnológicamente posible hoy en día. Una de las iniciativas de esta agencia fue la DARPA Grand Challenge, una carrera de vehículos autónomos en la que estos deben llegar desde un punto de los Estados Unidos hasta otro sin intervención humana y disponiendo únicamente de un listado de puntos intermedios entre el principio del circuito y el final. *DARPA* concede premios en metálico y en cada nueva edición es mayor el número de coches sin conductor que la completan y son capaces de desplazarse mayores distancias de forma más segura. En la Fig. 1.10 podemos ver la foto de los vehículos participantes en la edicion de 2007.

Figura 1.10 Fotos de los 6 finalistas del Urban Challenge de 2007

El objetivo del presente proyecto es indagar en la implementación de sensores en un coche, en este caso usando la plataforma ROS (Robot Operating System), que facilita la comunicación entre distintos periféricos y el ordenador principal de control. Se pretende implementar un GPS para detectar la posición del coche, un sensor inercial para analizar la dinámica del coche, y una cámara estéreo para conseguir detectar peatones.

En el próximo capítulo se explicarán los fundamentos en los que se basan los sensores utilizados. A continuación se especificará el software usado y su funcionamiento. Finalmente, se profundizará en el desarrollo del proyecto.

2. Fundamentos teóricos

En el presente capítulo se pretende introducir los conocimientos necesarios para poder entender un poco mejor el funcionamiento de los sensores y los procesos involucrados en el proyecto.

El capitulo empieza tratando los fundamentos en los que se basa la captura de imágenes, a continuación se ve el modelo usado para la detección de peatones, y por último se introducen los sensores usados en el proyecto y su principio de funcionamiento.

2.1 Principios ópticos

Las cámaras tienen la función de captar la información visual del entorno. Para ello, van equipadas con un sensor, normalmente un sensor tipo CCD(Charge Coupled Device,dispositivo de carga acoplada) o CMOS(Complementary Metal Oxide Semiconductor, semiconductor metal-óxido complementario). Para conseguir que los rayos de luz incidan sobre el sensor, se suele dotar a la cámara con una lente u óptica, cuya misión principal es precisamente esa, la de hacer llegar los rayos al sensor [15].

2.1.1 Modelos matemáticos de lente

Se tienen dos modelos matemáticos básicos que explican el comportamiento de una lente y modelos más complicados como el que se explica aquí [16]:

Modelo Pin-Hole: El modelo *pin-hole* es el modelo más básico, y simplifica la óptica a un único punto situado a una distancia focal de la imagen, como vemos en la Fig. 2.1. Este método es en el que se basaban las primeras cámaras. La información 3D del mundo se proyecta en el plano de la imagen a través del centro óptico situado a una distancia igual a la distancia focal del plano de la imagen. Esta distancia es el parámetro principal para calcular la posición y el tamaño de los objetos en la imagen. El plano focal es el plano que pasa por el centro óptico y cuyos puntos no tienen proyección en el plano de la imagen. Por supuesto el centro óptico esta contenido en el plano óptico.

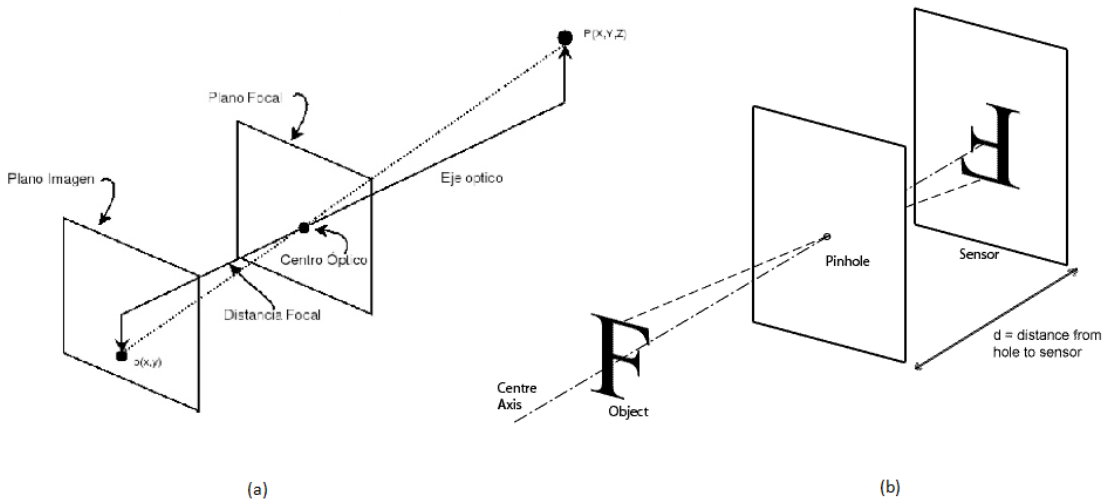


Figura 2.1 Modelo Pin-Hole (a) y ejemplo (b).

En este modelo, las ecuaciones que relacionan las coordenadas de un punto del mundo (X_c,Y_c,Z_c), con su proyección en la imagen (X,Y) son las ecuaciones (2.1) y (2.2). Tambien se puede ver visualmente esta relación en la Fig. 2.2:

$$X = f\frac{X_c}{Z_c} \qquad (2.1)$$

$$Y = f\frac{Y_c}{Z_c} \qquad (2.2)$$

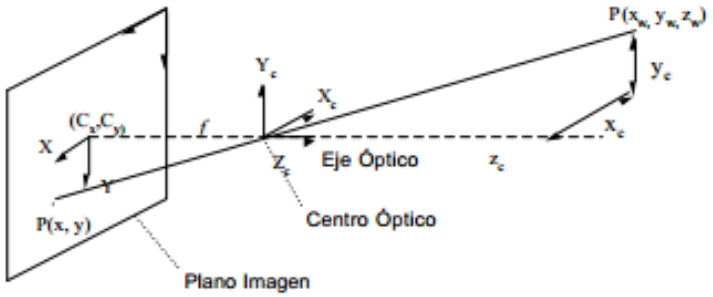


Figura 2.2 Relación entre el mundo y la proyección en el modelo Pin-Hole.

Modelo ideal o de lente fina: En el proceso de captación de una imagen, los rayos inciden desde diversos ángulos sobre la lente. Los rayos que proceden de un mismo punto, convergen a una distancia que depende de la distancia a la que se encuentre el punto real de la lente. Esta dependencia viene dada por la ecuación (2.3), que relaciona las distancias de un punto y su proyección con la distancia focal f.

$$\frac{1}{d_{in}} + \frac{1}{d_{out}} = \frac{1}{f} \qquad (2.3)$$

La trayectoria de la luz que atraviesa la lente por su centro de forma perpendicular no provoca ninguna distorsión y recibe el nombre de eje óptico. Los rayos que inciden de forma perpendicular sobre la lente, cortan al eje óptico a una distancia igual a la distancia focal como se puede ver en la Fig. 2.3.

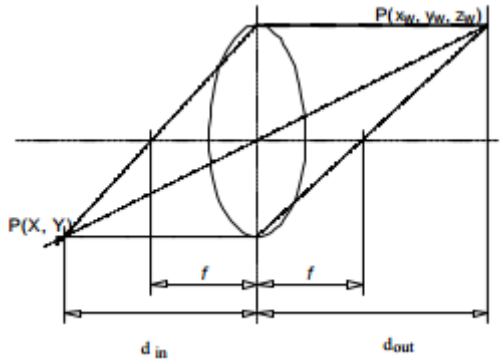


Figura 2.3 Modelo de lente fina.

Modelo de lente gruesa: Este modelo es un poco más complejo y tiene en cuenta otros parámetros al considerar que la lente tiene un cierto espesor. Esto da lugar a las aberraciones, que son los errores producidos por la lente utilizando este modelo. Para reducir las aberraciones que se producen en una lente, los sistemas de lentes constan de varios juegos de lentes coaxiales. Los puntos de referencia en este modelo son los puntos nodales, delantero y trasero, que sustituyen al centro óptico del modelo anterior, los puntos focales, que son dos puntos situados sobre el eje óptico y conjugados a puntos del infinito, y los puntos principales, que son dos puntos conjugados situados sobre el eje óptico, intersección sobre éste de los planos con magnificación unidad, es decir, los planos en los que $d_{in} = d_{out}$. Se puede ver una representación de este modelo en la Fig. 2.4.

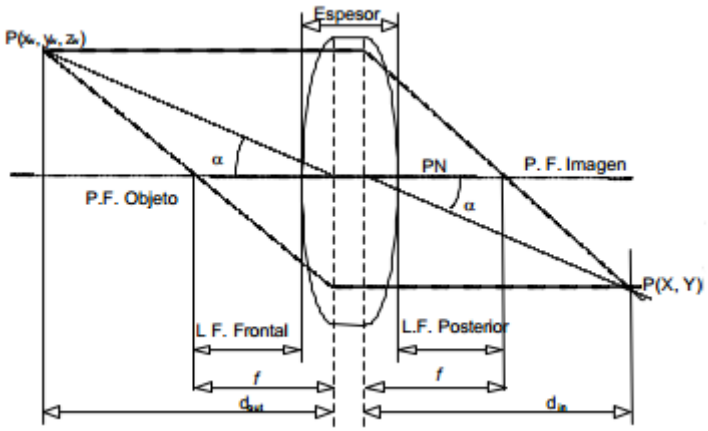


Figura 2.4 Modelo de lente gruesa.

2.1.2 Visión estereoscópica o estéreo

La visión estereoscópica constituye un procedimiento para la obtención de la forma de los objetos en la escena. En este caso la forma se determina a través de la distancia de los objetos en relación con un sistema de referencia por lo que se trata de un método para la obtención de la tercera dimensión.

Para la obtención de la estructura de la escena, existen dos tipos de métodos: activos y pasivos. Los métodos pasivos son aquellos que intervienen externamente sobre la escena, bien iluminándola o bien enviando un haz energético, tales como sensores de ultrasonidos, luz estructurada, triangulación, telémetro de tiempo de vuelo, etc. Por el contrario, los pasivos no actúan sobre la escena [18].

La visión estereoscópica toma como referencia el modelo estereoscópico biológico, donde el desplazamiento relativo de los ojos permite obtener la profundidad de los objetos o tercera dimensión mediante un simple proceso de triangulación a partir de las dos imágenes generadas por el mismo objeto de la escena 3D en cada ojo. Gracias al hecho de que los ojos están distanciados, se consigue que las imágenes de los objetos en sendos ojos se muestren desplazadas según la distancia de los objetos a los ojos. Si se solapan las imágenes obtenidas en ambos ojos se obtiene la imagen de la Fig. 2.5 (b) en la que se observa que la separación relativa entre las imágenes de los dos triángulos es menor que la separación relativa entre las imágenes de las estrellas. Este fenómeno se explica por el hecho de que la estrella en la escena 3-D se encuentra más próxima a los ojos que el triángulo. Estas separaciones relativas de los objetos en las imágenes obtenidas en cada ojo, es lo que se denomina disparidad.

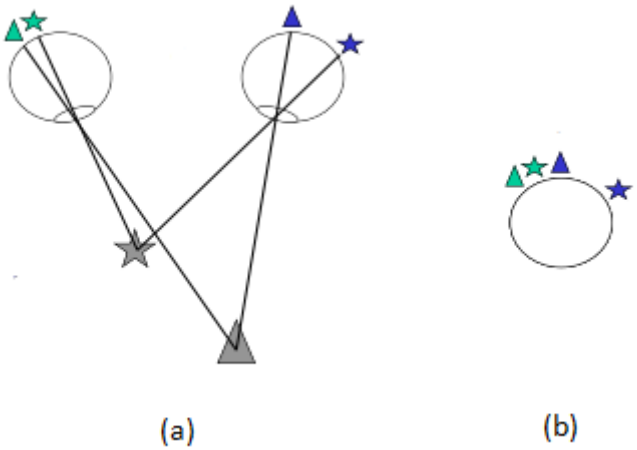


Figura 2.5 Sistema de visión estereoscópica biológico (a), y superposición de las imágenes de ambos ojos (b).

En visión estereoscópica artificial generalmente se utilizan dos cámaras separadas entre sí una cierta distancia relativa, con las que se obtienen las correspondientes imágenes del par estéreo. El procedimiento consiste en captar dos imágenes de una misma escena. Cada imagen es capturada desde una posición de las cámaras ligeramente diferente, por lo que las imágenes se presentan también ligeramente desplazadas entre sí, siendo éste el fundamento básico de la visión estereoscópica, ya que este hecho es el que va a permitir la obtención de la distancia a la que se encuentra un determinado objeto.

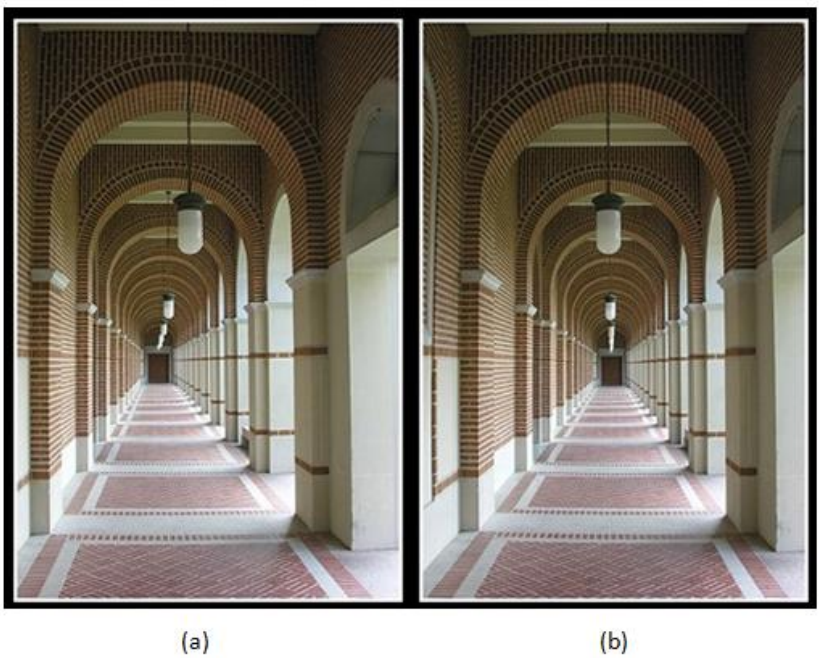


Figura 2.6 Par de imágenes estéreo: izquierda (a), y derecha (b).

En la Fig. 2.6 se muestra un par de imágenes estereoscópicas captadas mediante un sistema de visión artificial con dos cámaras alineadas horizontalmente, de forma que los objetos en las imágenes sólo presentan un desplazamiento horizontal y no vertical.

La captura de las imágenes de la escena se puede obtener, bien alineando dos cámaras de forma que se sitúen ligeramente desplazadas en el espacio, o bien, desplazando una única cámara una cierta distancia y captando las imágenes en las diferentes posiciones de desplazamiento.

En ambos casos la geometría del sistema puede diseñarse de forma que las cámaras tengan sus ejes ópticos paralelos o convergentes. El modelo más utilizado en visión artificial es el de ejes ópticos paralelos y es en el que se centrará la explicación.

Un sistema convencional está caracterizado por un par de cámaras, con sus ejes ópticos (Z_I y Z_D) mutuamente paralelos y separados por una distancia horizontal que se denomina línea base, y que se puede ver en la Fig. 2.7 identificada mediante el parámetro b.

Se denomina plano epipolar al formado por el punto *P de la imagen* y los centros ópticos de ambas cámaras, O_I y O_D . Las líneas epipolares son las formadas por la intersección del plano de cada una de las imágenes con el plano epipolar. Esto se puede ver gráficamente en la Fig. 2.8. Las cámaras tienen sus ejes ópticos perpendiculares a la línea base y sus líneas epipolares paralelas a la línea base.

Como se puede ver en la Fig, 2.7, en este sistema de ejes ópticos paralelos, el desplazamiento entre los centros ópticos de las dos cámaras es horizontal. Esto se traduce en el hecho de que las imágenes de un punto determinado de la escena captado por ambas cámaras difiere solamente en la componente horizontal.

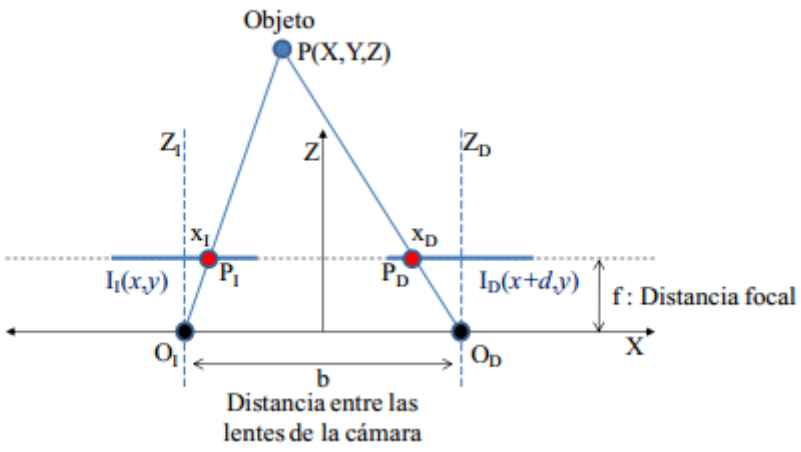


Figura 2.7 Representación de la proyección estéreo desde una perspectiva superior.

La Fig. 2.7 muestra la geometría de un par de cámaras en estéreo, representadas por sus modelos puntuales con sus planos imagen, I_I e I_D reflejados sobre sus centros de proyección, O_I y O_D, respectivamente.

El origen del sistema de coordenadas de referencia o del mundo está en O, punto que se encuentra en la intersección entre el eje X y el Z en la imagen, siendo la longitud focal efectiva de cada cámara f, y la línea base b como ya se ha comentado anteriormente. Como consecuencia de la geometría de la imagen, se obtiene la denominada restricción epipolar, que ayuda a limitar el espacio de búsqueda de correspondencias, de manera que en el sistema de ejes paralelos convencional, todos los planos epipolares originan líneas horizontales al cortarse con los planos de las imágenes. En un sistema con la geometría anterior se obtiene un valor de disparidad d, para cada par de puntos emparejados P_I (x_I, y_I) y P_D(x_D, y_D) dado por d = x_I - x_D, ya que y_I = y_D .

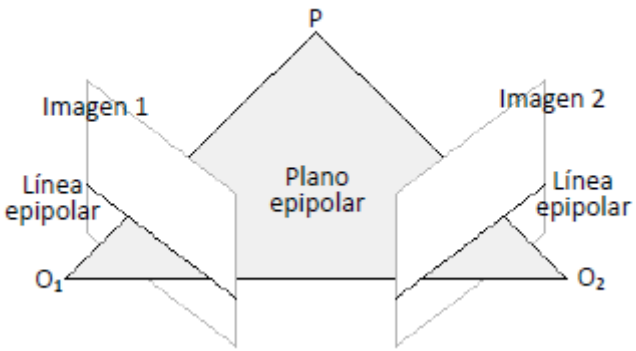


Figura 2.8 Geometría epipolar para una imagen normalizada.

Considerando una relación geométrica de semejanza de triángulos, las coordenadas del punto de la escena P(X,Y,Z) pueden deducirse fácilmente sin más que observar la Fig. 2.7, obteniéndose los resultados dados por la ecuación (2.4). Se deduce a partir de la ecuación (2.4) que cuando se utiliza esta geometría, la profundidad Z, es inversamente proporcional a la disparidad de la imagen y que para una profundidad dada, si tenemos un b mayor, tambien obtendremos un mayor d.

$$\left.\begin{array}{l} O_I: \frac{\frac{b}{2}+X}{Z} = \frac{x_I}{f} \\ O_D: -\frac{\frac{b}{2}-X}{Z} = \frac{x_D}{f} \end{array}\right\} \Rightarrow \left.\begin{array}{l} x_I = \frac{f}{Z}\left(X+\frac{b}{2}\right) \\ x_D = \frac{f}{Z}\left(X-\frac{b}{2}\right) \end{array}\right\} \Rightarrow d = x_I - x_D = \frac{fb}{Z} \Rightarrow Z = \frac{fb}{d} \qquad (2.4)$$

2.2 Histograma de gradientes orientados (HOG)

2.2.1 Introducción

El HOG (Histogram of Oriented Gradients, histograma de gradientes orientados) es un método usado para la detección de patrones en una imagen. El motivo por el que este método es tan usado, es porque el gradiente de la imagen en ambas direcciones, con sus correspondientes módulos y ángulos, es una buena herramienta para la distinción de distintos patrones [19].

Un sistema de reconocimiento de patrones completo consiste en un sensor que recoge imágenes que deben ser clasificadas, un sistema de extracción de características que transforma la información observada en valores numéricos o simbólicos, y un sistema de clasificación o descripción que, basado en las características extraidas, clasifica la medición.

Para la clasificación se puede usar un conjunto de aprendizaje, del cual ya se conoce la clasificación de la información a priori y el cual se usa para entrenar al sistema, siendo la estrategia resultante conocida como aprendizaje supervisado. El aprendizaje puede ser tambien no supervisado, lo que significa que el sistema no tiene un conjunto para aprender a clasificar la información a priori, y por lo tanto, se basa en cálculos estadísticos para clasificar los patrones.

El algoritmo Histograma de gradientes orientados se usa en ámbitos como la visión por computador y el procesamiento de imágenes, con el propósito de detectar objetos en una imagen. La esencia de dicho algoritmo es que la forma de un objeto en una imagen puede ser descrita por medio de la distribución de los gradientes. El objetivo principal de esta técnica es la extracción de características de una imagen. Las características son extraídas teniendo en cuenta los bordes. El proceso de obtener información de los bordes presentes en una imagen, se consigue calculando los gradientes y las orientaciones de los píxeles.

2.2.2 Fases del proceso del HOG

El proceso llevado a cabo para conseguir la detección mediante este método es el siguiente [20]:

En primer lugar, la imagen tiene que pasar por un preproceso para aumentar la eficiencia y eficacia de las fases posteriores. En esta fase, lo que se precisa conseguir es transformar una imagen en color, a otra en escala de grises.

A continuación, se tiene que calcular el gradiente de la imagen en las dos direcciones, x e y. En este método se calcula el gradiente mediante la convolución con filtros gaussianos de distinta varianza.

A partir de estos dos gradientes, se calculan la magnitud y la orientación del gradiente mediante las siguientes fórmulas (2.5) para la magnitud y (2.6) para la orientación.

$$|G| = \sqrt{I_x^2 + I_y^2} \tag{2.5}$$

$$\theta = arctan\left(\frac{I_x}{I_y}\right) \tag{2.6}$$

El siguiente paso consiste en dividir la imagen en celdas y calcular el gradiente de cada una. Esta división se hace de forma que todas las celdas sean iguales y el número de píxeles de cada una en la dirección x y en la y sea el mismo. A partir de esta división, mas adelante se podrán identificar las regiones de interés, que en este caso serán las partes de la imagen que pueden contener a un peatón. Dentro de una celda, cada píxel tiene una cierta magnitud y una cierta orientación, las cuales fueron calculadas previamente. Lo que se va a hacer es dividir los posibles valores de orientación que pueden tomar los píxeles de la imagen en varios rangos. Para cada celda, se almacena el valor de la magnitud de un píxel según la orientación que este tiene, con lo que conseguimos formar para cada celda un histograma de orientación. Una imagen se puede representar por medio del histograma de cada una de las celdas, los cuales se ordenan en un vector según su peso, dando lugar al vector de características de la imagen. Un caso particular de vector de características es el de una imagen omnidireccional, es decir, una imagen que tiene los mismos píxeles en una fila aunque este rotada, ya que el vector de características es invariante ante la rotación.

Al tener que realizar el proceso para cada celda, este método se vuelve bastante lento. Se pueden ver las fases de procesado nombradas en la Fig. 2.9.

Una ventaja de este método es su gran robustez frente a cambios en la iluminación y frente a pequeños cambios en el contorno, así como a cambios en la escala y en el fondo.

Este proyecto no se centra en la velocidad de detección, por ello no será tan importante la eficiencia de computación.

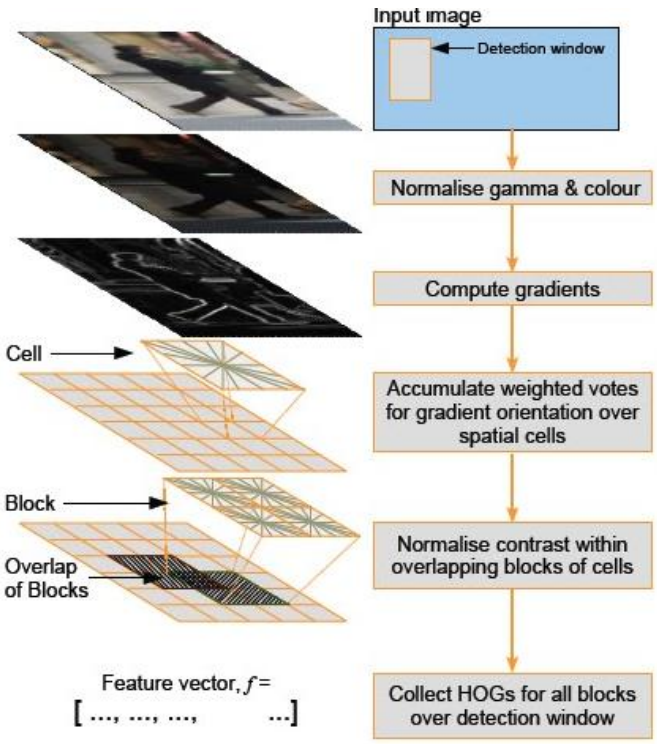


Figura 2.9 Fases de procesado en el HOG.

Si se necesitara aumentar la velocidad de procesamiento, se podría recurrir a utilizar la técnica denominada integral de HOG, la cual mejora el tiempo de procesamiento a costa de aumentar la memoria necesaria para llevarlo a cabo. Para ello se necesitan n matrices auxiliares de tamaño igual a la imagen de entrada WxH, siendo W y H el ancho y alto respectivamente, y n el número de clases en el HOG. Cada una de las matrices estará asociada a una clase, es decir a un único rango de ángulos del gradiente, y donde cada píxel cumple la ecuación (2.7).

$$M(b,i,j) = \begin{cases} |G(i,j)|, \theta(i,j), & \epsilon \, \mathrm{R}(b) \\ 0, & eoc \end{cases} \qquad (2.7)$$

La integral de HOG está compuesta de n matrices de tamaño WxH, cada una de ellas asociadas a una clase y donde cada píxel (i, j) indica la suma de las magnitudes del rectángulo de la imagen, siendo la esquina superior izquierda el píxel (0,0) y con un tamaño de ixj. Estas matrices se calculan a partir de las matrices M anteriores y se puede definir una integral de HOG como la función recursiva de la ecuación (2.8).

$$IH(b,i,j) = \begin{cases} 0, & i = 0 \; \forall \; j = 0 \\ IH(b,i,j-1) + \sum_{k=0}^{i} B(b,k,j), & eoc \end{cases} \quad (2.8)$$

De esta forma, para el cálculo del HOG de cada celda de un descriptor de HOG se deberá realizar una simple resta por cada clase por medio de la siguiente ecuación (2.9).

$$C(b,i,j,W_c,H_c) = IH(b,i+W_C,j+H_C) - IH(b,i+W_c,j) - \quad (2.9)$$
$$-IH(b,i,j+H_c) + IH(b,i,j)$$

En la Fig. 2.10 se puede ver gráficamente el proceso ejecutado para realizar la integral de HOG.

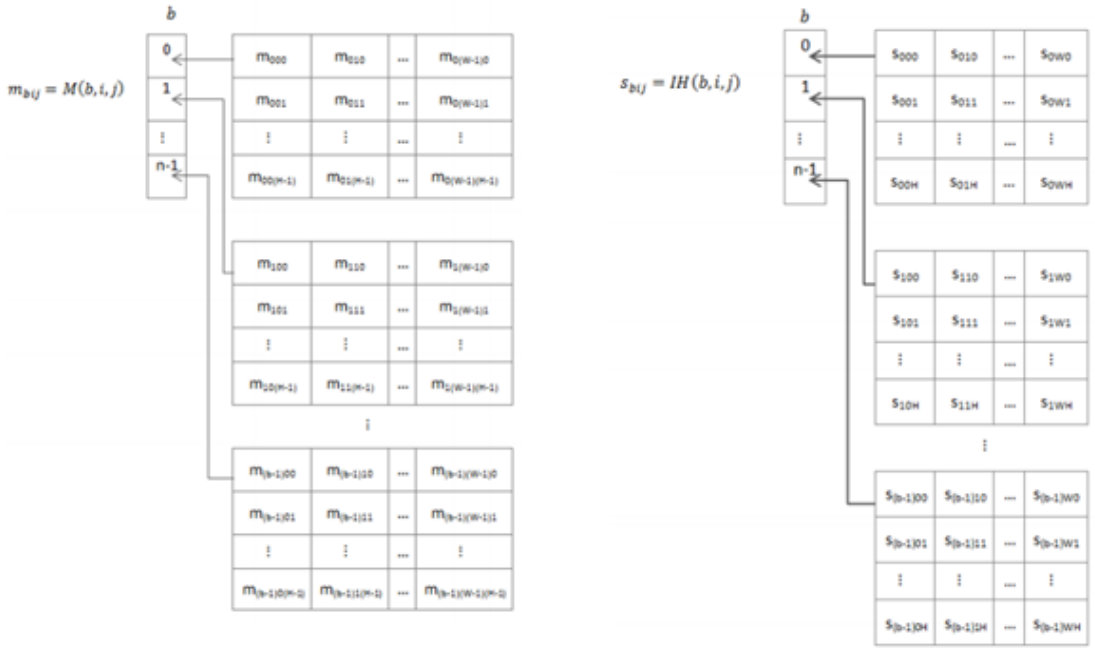


Figura 2.10 Matrices auxiliares y esquema de la integral de HOG.

2.2.3 Histograma de gradientes orientados para peatones

Para la implementación del programa de detección de peatones realizado en ROS se parte del ejemplo "people detect" incluido en las librerias de OpenCV y mostrado en la sección de programas y código. Este se basa en la recopilación de información de una gran cantidad de imágenes separadas en dos grupos, las que contienen figuras humanas (positivas), y las que no (negativas), con el objetivo de entrenar un detector mediante la técnica denominada Support Vector Machine (SVM). Cada imagen dará lugar a un histograma de gradientes orientados, con lo que se podrá formar un descriptor de histogramas que servirá para poder comparar con los histogramas obtenidos de las imágenes a analizar.

La técnica de SVM es un método de aprendizaje supervisado que, a partir de unas muestras de entrada, reconoce patrones y permite resolver problemas de clasificación y de regresión. En este caso, este método se usa para identificar peatones. Se trata de un problema de identificación de dos clases: una clase asociada a los peatones y otra a todo lo que no lo sea.

Se puede representar el problema de clasificación en dos clases de una forma abstracta. Partiendo de un conjunto de muestras iniciales, unas positivas y otras negativas, con n características discriminantes cada una, se posiciona cada muestra como un punto en el espacio de n dimensiones. El objetivo es encontrar, si es posible, un hiperplano que separe los puntos de ambas clases en dos grupos, intentando que la distancia entre las dos clases sea la mayor posible para separarlas más fácilmente y obtener una mejor clasificación. En la Fig. 2.11 se muestra el caso en el que n=2, es decir, se tienen dos características.

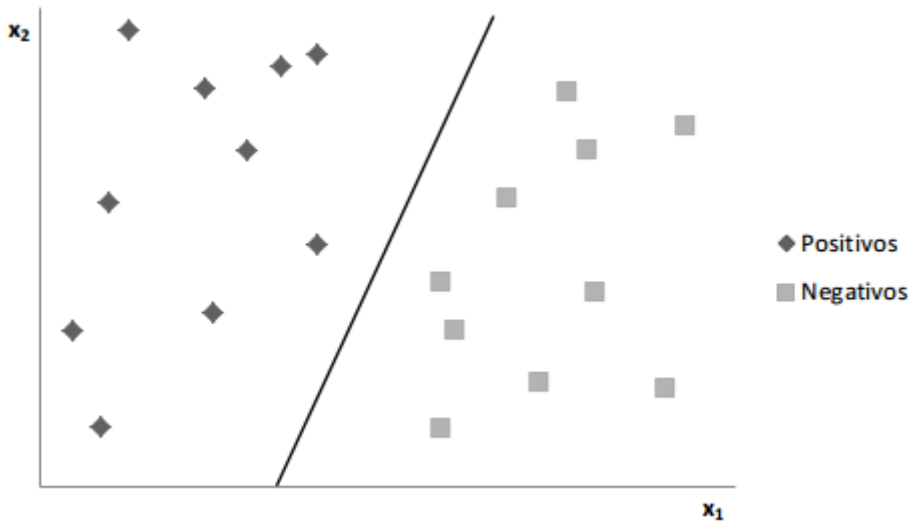


Figura 2.11 Hiperplano de separación de dos clases.

La descripción matemática de este método para el caso de tener dos características es la siguiente:

Dada una entrada compuesta por M muestras de entrenamiento $x_i \in \mathbb{R}^n (i = 1,\ldots,M)$ pertenecientes a las clases C_1 o C_2 y las etiquetas asociadas y_i {1,-1} siendo $y_i = 1$ si x_i pertenece a la clase C_1 e $y_i = -1$ si pertenece a la clase C_2, se dice que las M muestras son linealmente separables si podemos determinar su función de decisión mediante la ecuación (2.10) donde $w \in \mathbb{R}^n$, $b \in \mathbb{R}$.

$$D(x) = w\Delta x + b \tag{2.10}$$

Para i = 1,…,M se cumple la ecuación (2.11).

$$w\Delta x_i + b \begin{cases} > 0 \; para \; y_i = 1 \\ < 0 \; para \; y_i = -1 \end{cases} \tag{2.11}$$

Suponiendo que los datos de entrada son linealmente separables, ninguna entrada x_i satisface $w\Delta x_i + b = 0$. Para controlar la separabilidad entre las clases y poder separar las muestras en función del valor de su etiqueta, se sustituyen las anteriores desigualdades por las de la ecuación (2.12).

$$w\Delta x_i + b \begin{cases} \geq 1 \; para \; y_i = 1 \\ \leq -1 \; para \; y_i = -1 \end{cases} \tag{2.12}$$

Que se puede simplificar, quedando como aparece en la ecuación (2.13).

$$y_i(w\Delta x_i + b) \geq 1 \; para \; i = 1, \ldots, M \tag{2.13}$$

$$D(x) = w\Delta x + b = c \; para - 1 < c < 1 \tag{2.14}$$

El hiperplano de la ecuación (2.14) separa los datos de entrada. Si c = 0 este hiperplano se encuentra en medio de los hiperplanos con c = 1 y c= -1. El margen es la distancia existente entre el hiperplano de separación y la muestra de entrenamiento más cercana al mismo. Suponiendo que los hiperplanos D(x) = 1 y

D(x) = -1 contienen al menos una muestra cada uno, D(x) = 0 tiene el máximo margen para -1< c < 1. Este se denomina hiperplano óptimo de separación.

El vector w es ortogonal al hiperplano de separación, por lo tanto, la linea que forman una muestra de entrada x y el vector w es tambien ortogonal a la misma. Esta linea esta definida por la ecuación (2.15), donde |d| es la distancia entre x y el hiperplano. Igualando a cero y despejando se obtiene el valor de la distancia, como se muestra en la ecuación (2.16).

$$\frac{d}{\|w\|}\Delta w + x \qquad (2.15)$$

$$d = -\frac{D(x)}{\|w\|} \qquad (2.16)$$

Para los hiperplanos D(x) = 1 y D(x) = -1 la distancia es $||w||^{-1}$ en ambos casos, por lo que la distancia entre ellos es $2||w||^{-1}$. En la Fig. 2.12 se pueden ver estas distancias con el hiperplano óptimo de separación.

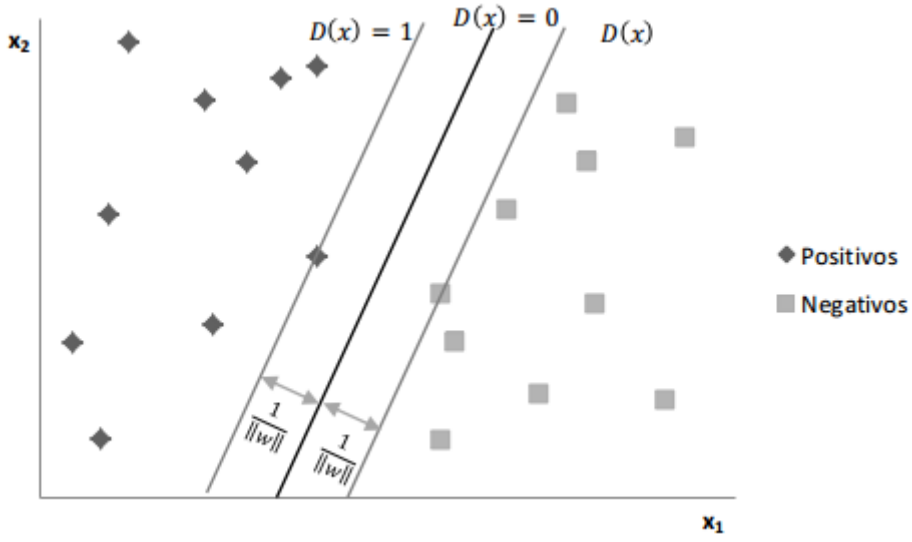


Figura 2.12 Hiperplano de separación óptimo y sus distancias de margen.

El objetivo será maximizar esta distancia para obtener una mejor discriminación entre clases. El problema es el mostrado en la ecuación (2.17).

$$maximizar\ f(w,b) = \frac{2}{\|w\|}\ tal\ que\ y_i(w\Delta x + b)\ \geq 1 para\ i = 1, \dots, M \quad (2.17)$$

Que se puede transformar en un problema de programación cuadrática cambiando la maximización por una minimización como la de la ecuación (2.18).

$$minimizar\ g(w,b) = \frac{\|w\|^2}{2}\ tal\ que\ y_i(w\Delta x + b)\ \geq 1 para\ i = 1, \dots, M \quad (2.18)$$

El valor de la solución objetivo es único. Usando el método de multiplicadores de Lagrange se convierte la expresión con restricciones en una que no las tiene, como se puede ver en la ecuación (2.19) donde $\alpha = (\alpha_1,\dots,\alpha_M)$ y α_i es un multiplicador de Lagrange no negativo.

$$h(w,b,\alpha) = \frac{1}{2}\Delta\|w\|^2 - \sum_{i=1}^{M} \alpha_i y_i\ (w\Delta x_i + b) - 1 == \frac{1}{2}\Delta\|w\|^2 - \sum_{i=1}^{M} \alpha_i \Delta y_i\ (w\Delta x_i + b) + \sum_{i=1}^{M} \alpha_i \quad (2.19)$$

Se obtiene la solución óptima cuando se minimiza la ecuación respecto a w, se maximiza respecto a α_i, y se maximiza o minimiza respecto a b, segun el signo del sumatorio de $\alpha_i \Delta y_i$.

Se utiliza el método de Kuhn-Tucker para resolver y ver si la solución es óptima. Haciendo esto se obtienen las siguientes ecuaciones.

$$\frac{\partial h(w,b,\alpha)}{\partial w} = 0 \quad (2.20)$$

$$\frac{\partial h(w,b,\alpha)}{\partial b} = 0 \quad (2.21)$$

$$\alpha_i y_i(w\Delta x_i + b) - 1 = 0\ para\ i = 1, \dots, M \quad (2.22)$$

$$\alpha_i \geq 0\ para\ i = 1, \dots, M \quad (2.23)$$

De las ecuaciones anteriores se tiene que, o $\alpha_i = 0$, o $\alpha_i \neq 0$ e $y_i(w\Delta x_i + b) = 1$. Las muestras con $\alpha_i \neq 0$ son los vectores de soporte.

Si se siguen desarrollando las ecuaciones llegamos a estas nuevas ecuaciones:

$$\frac{\partial h(w,b,\alpha)}{\partial w} = w - \sum_{i=1}^{M} \alpha_i \Delta y_i\ \Delta x_i = 0 \Rightarrow$$

$$\Rightarrow w = \sum_{i=1}^{M} \alpha_i \Delta y_i\ \Delta x_i \quad (2.24)$$

$$\frac{\partial h(w,b,\alpha)}{\partial b} = w - \sum_{i=1}^{M} \alpha_i \Delta y_i = 0 \quad (2.25)$$

Ahora, sustituyendo (2.24) y (2.25) en (2.19), se obtiene la ecuación (2.26).

$$h(w,b,\alpha) = \frac{1}{2}\Delta \sum_{i=1}^{M} \alpha_i \Delta y_i \, \Delta x_i \Delta \sum_{j=1}^{M} \alpha_j \Delta y_j \, \Delta x_j -$$

$$- \sum_{i=1}^{M} \alpha_i \Delta y_i \, \Delta\left(\sum_{j=1}^{M} \alpha_j \Delta y_j \, \Delta x_j \Delta x_i + b\right) + \sum_{i=1}^{M} \alpha_i \quad (2.26)$$

El problema que hay que resolver se transforma entonces en el de la ecuación (2.27).

$$maximizar\ h(\alpha) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2}\Delta \sum_{i=1}^{M} \alpha_i \Delta \alpha_j \Delta y_i \, \Delta y_j \Delta x_i \Delta x_j$$

$$tal\ que \sum_{i=1}^{M} \alpha_i \Delta y_i = 0,\ \alpha_i \geq 0\ para\ i = 1, \ldots, M \qquad (2.27)$$

Si se resuelve este problema, se obtiene una solución para cada multiplicador de Lagrange α_i, y las muestras x_i asociadas a un α_i positivo serán los vectores de soporte.

La función de decisión queda como se ve en la ecuación (2.28), siendo S el conjunto de índices de las muestras que son vectores de soporte.

$$D(x) = \sum_{i \in S} \alpha_i \Delta y_i \Delta x_i \, \Delta x + b \qquad (2.28)$$

A partir de esta ecuación y de la ecuación (2.25) se obtienen las ecuaciones (2.29) y (2.30).

$$y_i(w\Delta x_i + b) = 1 \qquad (2.29)$$

$$b = \frac{1}{y_i} - w\Delta x_i = y_i - w\Delta x_i \qquad (2.30)$$

Para tener mayor precisión, es mejor tomar una media aritmética sobre los vectores de soporte para el valor de b, tal y como se muestra en la ecuación (2.31).

$$b = \frac{1}{|S|} \sum_{i \in S} y_i - w\Delta x_i \qquad (2.31)$$

Ahora ya se pueden clasificar las muestras mediante la función de clasificación mediante el criterio de (2.32).

$$\begin{cases} Clase\ C1 & si\ \ D(x) > 0 \\ Clase\ C2 & si\ \ D(x) < 0 \end{cases} \tag{2.32}$$

Si se tiene D(x) = 0, entonces no se puede clasificar la muestra, ya que se encuentra en la frontera entre las dos clases.

Si los datos de entrada no pudieran ser separados linealmente, este método no daría buenos resultados. En ese caso, se podría añadir una cierta compensación entre la maximización del margen y la minimización del error de clasificación para solucionar el problema. A la compensación añadida en ese caso se le llama margen C.

Para el entrenamiento del SVM se toman dos conjuntos de datos. En el caso del ejemplo "people detect", el conjunto de datos de peatones MIT, contiene 509 entrenadores y 200 imágenes de prueba con peatones en distintas escenas de una ciudad. Estas imágenes contienen vistas con un número limitado de posturas de los peatones y todas con vistas delanteras o traseras de los mismos, lo cual da buenos resultados, pero mejorables. Para mejorar los resultados, se recurre al segundo conjunto de datos, denominado INRIA y que contiene 1805 imágenes de personas de 64x128 en fondos y con posturas muy variadas, apareciendo incluso en algunas, grupos de personas juntas. El proceso de aprendizaje es el siguiente: una vez que se tiene el primer modelo de detección explicado anteriormente, se compara con las imágenes de entrenamiento. Si una imagen entrenada como positiva no se detecta, se tiene un falso negativo, y si una imagen que es negativa se detecta como positiva, se tiene un falso positivo. A continuación se incorporan en el modelo los falsos negativos y positivos detectados, al grupo adecuado. Se repite este proceso hasta obtener una precisión suficiente del detector.

Una vez entrenado el detector, se divide la imagen a clasificar en celdas, calculando el histograma de orientación de los bordes en cada una de ellas. El histograma divide el rango de valores posibles de los píxeles en una serie de clases de igual o distinto tamaño. En cada clase se almacena la frecuencia de píxeles con un valor comprendido dentro del subrango de esa clase. Igualmente el histograma de gradientes orientados de una imagen divide estas clases segun la orientación de los gradientes de los píxeles, es decir, el ángulo que toma. Para dividir la orientación en subclases, lo mejor es tomar el rango [0º, 180º] y dividirlo en 9 clases iguales, [0º, 20º]... [160º, 180º], con lo que en cada clase se almacenará el número de píxeles cuyos gradientes esten orientados con un ángulo comprendido en el intervalo de

esa clase. Para mejorar la invariancia ante factores externos como la iluminación, se puede normalizar el contraste de los histogramas de cada una de las celdas en las que se divide la ROI (Region Of Interest).

Correspondencia entre dos conjuntos de descriptores

Si se quiere agilizar el proceso de detección de peatones, se puede estimar la posición del peatón en una imagen posterior a la actual, siguiendo para ello los pasos detallados a continuación:

Una vez que se tienen determinados los descriptores HOG que representan la región a detectar, en este caso el peatón, se puede establecer una correspondencia entre el peatón y las posibles ubicaciones que puede ocupar este en una imagen posterior. Para ello, se implementan dos métricas y se obtiene una medida de disimilitud entre ambas imágenes. Estas métricas son: el error cuadrático medio entre todas las celdas que representan al peatón y las celdas de cada una de las posibles ubicaciones del peatón en la imagen posterior; y el error cuadrático medio sobre un porcentaje del número de celdas total, las cuales son elegidas por presentar los mínimos errores.

Error cuadrático medio de todo el conjunto

Para realizar el cálculo del error cuadrático medio se necesita obtener previamente los dos conjuntos de descriptores. Para las fórmulas, se denominarán descriptores A a los de la primera imagen o imagen A, y descriptores B a los de la segunda o imagen B.

Teniendo en cuenta que para una imagen el número de descriptores total(TD) es el obtenido de la ecuación (2.33), siendo NC el número de celdas en que se divide la imagen. Se tiene, por tanto, que el error cuadrático medio(E_{cm}) es el que aparece en la ecuación (2.34) siendo DA y DB los vectores de características de cada imagen.

$$TD = NCx9 \tag{2.33}$$

$$E_{cm} = \frac{1}{TD}\Delta \sum_{i=0}^{TD}(DA_i - DB_i)^2 \tag{2.34}$$

Este error obtenido es inversamente proporcional a la similitud entre las dos imágenes.

Error cuadrático medio de un porcentaje

Este error es muy parecido al descrito anteriormente. La diferencia entre uno y otro es que en el segundo solo se tienen en cuenta un porcentaje del total de los descriptores elegido por el usuario. Lo que se pretende con esta métrica es buscar otra medida de correspondencia que aporte solo información de las celdas que contienen los mínimos errores.

Para calcularlo, se necesita crear un vector que contenga la potencia de los errores entre los elementos i-ésimos de cada uno de los vectores, de forma que la fórmula queda como se puede ver en la ecuación (2.35).

$$E_i = (DA_i - DB_i)^2 \quad i = 1,2,\dots,TD \tag{2.35}$$

A continuación, se ordena el vector E_i de manera ascendente para obtener el vector E_i^*, con el cual se calculará la nueva métrica de error. Si se nombra al porcentaje de puntos que serán incluidos en la medida como P, el error cuadrático medio porcentual será el que aparece en la ecuación (2.36).

$$E_{cm}^{\%} = \frac{1}{TC \times P} \Delta \sum_{i=1}^{TC \times P} E_i^* \tag{2.36}$$

Este número puede tomar valores entre 0 y 1, correspondiendo los valores más bajos con los casos en los que las imágenes son más similares entre si.

Una vez que se tiene todo lo anterior calculado se procede con el algoritmo para la detección de peatones.

Algoritmo de seguimiento de peatones

El algoritmo de seguimiento se basa en una ventana de búsqueda.

Después de obtener los descriptores del peatón en la imagen actual, se abre una ventana de búsqueda de M x M píxeles en la imagen siguiente. Esta ventana se centra en la posición inicial del peatón, y su tamaño es definido mediante M por el usuario, con la condición de que sea impar. Con esta ventana se pretende limitar el espacio de búsqueda para el cálculo de correspondencia entre la imagen anterior y la actual. Cada correspondencia es guardada en una matriz de error, de igual tamaño al de la ventana, que representa las posibles nuevas ubicaciones del peatón en la imagen actual.

A partir de las métricas de error calculadas anteriormente, se establece la correspondencia entre los descriptores del peatón y sus posibles ubicaciones en una

imagen posterior. Se implementa el algoritmo de seguimiento para cada una de las métricas.

Sobre la matriz de error se encuentra la correspondencia más alta o menor error para determinar la posición del peatón en la imagen siguiente. Con la nueva posición, además es posible determinar la trayectoria y distancia recorrida por el peatón, asi como la dirección de movimiento.

2.2.4 Histograma de gradientes orientados comprimido

En algunas aplicaciones, la velocidad del procesado de la imagen, y el tamaño de los vectores de características son clave, por ello el método anteriormente descrito no es el más adecuado.

Para mejorarlo, se comprimen los histogramas obtenidos usando el método de codificación de Huffman que se basa en la probabilidad de aparecer de cada valor [21]. Un ejemplo sencillo para entender este método de codificación es el siguiente:

Si se tiene la cadena {aabaacc}, la cual usa tres símbolos, "a", "b" y "c", con la probabilidad de que "a" aparezca más que los otros dos símbolos, lo que se hace es juntar primero los símbolos con menor probabilidad de aparecer como en la Fig. 2.13.

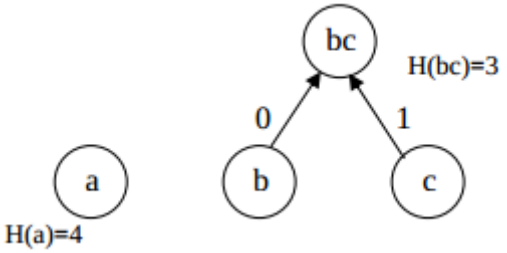


Figura 2.13 Codificación Huffman 1.

A continuación se une con el símbolo restante de la cadena, obteniendo el árbol de codificación final, como el que se puede ver en la Fig. 2.14.

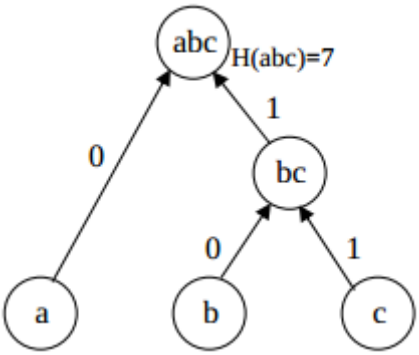


Figura 2.14 Codificación Huffman 2.

Para obtener el código de codificación de cada símbolo se recorre el árbol de arriba hacia abajo, de modo que "a" tendría el código "0", "b" el "10", y "c" el "11". La codificación de esta cadena sería "0 0 10 0 0 11 11", es decir, que se consigue representar la cadena con 10 bits en vez de los 14 que se necesitarían sin usar la codificación Huffman.

2.2.5 Histograma de gradientes orientados para video

Si se tuviera que utilizar un algoritmo para clasificar un video según sus características, el método anterior podría servir realizando pequeñas modificaciones sobre el mismo [22]. Para empezar, ahora no se tiene un problema de dos dimensiones, sino de tres, ya que el tiempo aparece como nueva dimensión. Las variables van a ser las variables espaciales "x" e "y" y la temporal "t". En el caso de dos dimensiones se calculaban la magnitud y el ángulo para los píxeles. Ahora se necesitarán dos ángulos en vez de uno, los cuales serán θ para el ángulo espacial, y ψ para el ángulo temporal. Al igual que antes, se divide el video en celdas, de manera que se dispone de imágenes separadas, las cuales, a su vez, se dividen de igual manera a como lo hacían en el caso de dos dimensiones (estas partes en las que se divide la imagen se clasificarán calculando el módulo y el ángulo de su gradiente, teniendo en cuenta que haber más de una imagen, habrá que añadir un ángulo extra para poder diferenciar entre distintas imágenes del video). Dependiendo del tiempo que corresponda a una imagen, su ángulo ψ tomará valores distintos. Se puede ver este esquema de celdas en la Fig. 2.15.

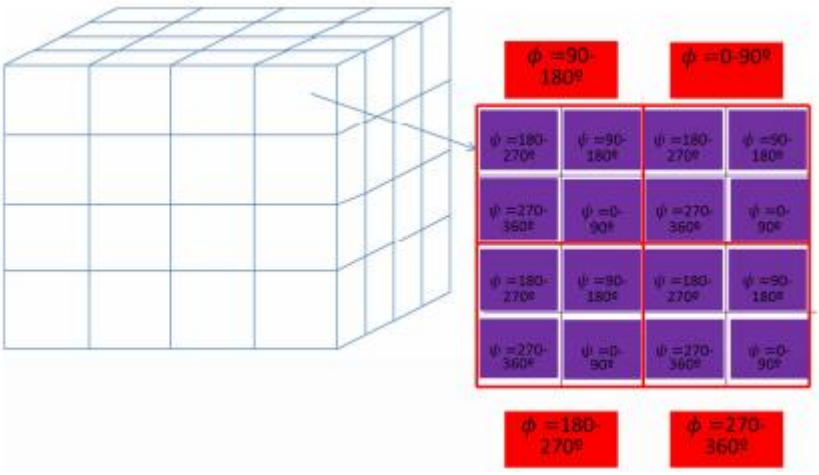


Figura 2.15 División del video en celdillas.

El algoritmo de clasificación comienza con la división del video en sus fotogramas. A continuación se escala cada imagen un factor a determinar para eliminar detalles innecesarios que no aportan información. Por ejemplo, en una imagen de un portátil, solo interesa el gradiente de los bordes del portátil y no el que se produce por la imagen en la pantalla del portátil. El siguiente paso es utilizar una compresión gamma (ɣ), con la que se elimina en gran parte el efecto de los pequeños detalles que no nos interesan sobre el gradiente y que aún se encuentran contenidos en la imagen. Como esta compresión reduce el gradiente, se debe hacer un reescalamiento del valor de los píxeles para que ocupen todo el intervalo de valores en el que se trabaja (de 0 a 255). Este reescalamiento se puede llevar a cabo mediante la ecuación (2.37), donde I^{γ} es el valor de los píxeles después de la compresión, y $max(I^{\gamma})$ es el mayor valor de todos los obtenidos.

$$I = 255\Delta\frac{I^{\gamma}}{max(I^{\gamma})} \tag{2.37}$$

Después se realiza el cálculo de los gradientes respecto a cada una de las tres coordenadas. Para ello se usan las siguientes máscaras derivativas centradas:

$$m_x = [-1 \quad 0 \quad 1] \tag{2.38}$$

$$m_y = [1 \quad 0 \quad -1]^T \tag{2.39}$$

$$m_z = [-1 \quad 0 \quad 1] \tag{2.40}$$

Con estas máscaras, el gradiente en cada dirección se calcula como aparece en la ecuación (2.41)(haciendo el producto vectorial de la matriz de los píxeles por la matriz formada por las máscaras).

$$G_{x,y,t} = I_{x,y,t} \otimes m_{x,y,t} \tag{2.41}$$

A partir de estos gradientes, el módulo y los ángulos se calculan con las siguientes fórmulas, (2.42) para el módulo, (2.43) para el ángulo espacial y (2.44) para el ángulo temporal:

$$|G| = \sqrt{G_x^2 + G_y^2 + G_t^2} \tag{2.42}$$

$$\theta = atan2(G_y, G_x) \tag{2.43}$$

$$\psi = atan2(G_x, G_t) \tag{2.44}$$

Una vez hecho esto, se divide el video video en celdas cúbicas, creando rangos de valores para los posibles valores de los ángulos y, como en el caso anterior, se van a sumar los módulos de los píxeles que tengan el valor de los ángulos pertenecientes a un mismo subrango. Si en el caso 2D se tenían 8 subrangos para ángulos desde 0º a 180º, ahora, al haber dos ángulos distintos, se tendrá el cuadrado de este valor, es decir, 8x8 = 64 subrangos.

Se procede a una normalización de los datos para tener todas las celdas en una escala común.

Finalmente, se forma el descriptor concatenando cada uno de los histogramas de los 64 subrangos angulares de cada una de las celdas. Esto crea un descriptor, que pude llegar a ser muy grande. Por ejemplo, en el caso de tener 4 celdas en las dos direcciones, 4 celdas temporales y 64 posibles combinaciones de ángulos, la longitud del descriptor será 4x4x4x64 = 4096. Para terminar, se realiza una normalización unitaria sobre el descriptor completo.

2.3 Sensores

2.3.1 Cámara estéreo

Las cámaras estereoscópicas se basan en el principio de la visión humana. Los ojos humanos estan separados unos 65 mm. entre sí. Cada ojo ve una imagen, que aún siendo parecidas entre sí, tienen distintos ángulos. El cerebro se encarga de mezclar las dos imágenes creando el efecto de profundidad. Esto lo consigue aprovechando la propiedad de que dependiendo de la distancia a la que se encuentre un objeto, este aparecerá con una mayor separación relativa entre las imágenes de cada ojo.

Así, capturando dos imágenes con una cierta separación entre ellas, se pueden imitar los ojos humanos. Si después, mediante el correcto procesado, se tratan estas imágenes, se podrá analizar analizar información 3D contenida en la imagen.

La captura de las dos imágenes se puede realizar de tres maneras: mediante una cámara especial con dos objetivos, mediante dos cámaras iguales con la captura de imágenes sincronizada, o mediante una cámara que se desplaza para tomar las dos imágenes. El que se usa en este proyecto es el segundo, que es el más extendido y el que permite obtener mejores resultados para distancias a partir de 1,5 m. En la Fig. 2.16 se puede ver la cámara usada en el proyecto, modelo Bumblebee 2 de la marca Point Grey.

Figura 2.16 Cámara estéreo usada en el proyecto.

2.3.2 Sensor inercial

Un sensor inercial es un sensor que mide aceleración y velocidad angular. Está compuesto por acelerómetros, giróscopos y magnetómetros. Los acelerómetros miden la aceleración lineal con que se mueve el sensor, los giróscopos, la velocidad angular, y los magnetómetros dan información acerca del norte magnético. Con estos tres sensores es posible estudiar el movimiento completo del sensor inercial en el

plano o el espacio, dependiendo de los ejes que posean los sensores. En la Fig. 2.17 se pueden ver los movimientos que son detectados por un sensor inercial. El modelo de GPS utilizado en este proyecto es un 3dmgx2 de la marca Microstrain como el de la Fig. 2.18.

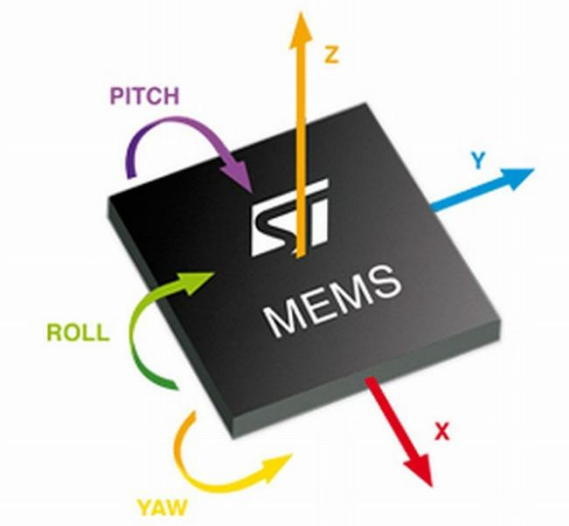


Figura 2.17 Movimientos de un sensor inercial.

Figura 2. 18 Sensor inercial usado en el proyecto.

2.3.3 GPS (Global Positioning System)

El GPS permite determinar en todo el mundo la posición de un objeto o persona con una precisión incluso de centímetros segun el tipo de GPS, aunque lo normal es tener una precisión de unos pocos metros. Esta formado por una red de 24 satélites y utiliza el método de triangulación para determinar la posición en el globo. Cuando se quiere saber la posición, el receptor envía una señal a un mínimo de tres satélites, los

cuales devuelven la identificación y la hora de cada uno. Con base a estas, el aparato sincroniza la hora y calcula el tiempo que tardan en llegar las señales. A partir de este tiempo se calcula la distancia a cada uno de los satélites, y por el método de triangulación y sabiendo las coordenadas de cada uno de los satélites, se obtiene la posición del dispositivo GPS.

Obviamente las señales obtenidas del GPS contienen errores debidos a diversos motivos. Un tipo especial de GPS es el GPS diferencial o DGPS. Es un sistema que proporciona a los receptores de GPS correcciones a las señales recibidas de los satélites GPS, con el fin de aumentar la precisión de la posición calculada. Este sistema se basa en el hecho de que los errores producidos por el sistema GPS afectan por igual o de forma similar a los receptores próximos entre sí. Si se tiene un receptor en tierra del cual se conoce su posición de antemano, se puede comparar el valor de posición real de este con el obtenido a partir de los satélites para calcular los errores producidos, y a partir de estos, modificar las posiciones obtenidas para los receptores que se encuentren en el rango del receptor de posición conocida. Es decir, además de los satélites, el receptor debe recibir la señal de una estación monitorizada de la cual sabemos la posición, para corregir parte de los errores producidos en la medición. Se puede observar el proceso en la Fig. 2.19.

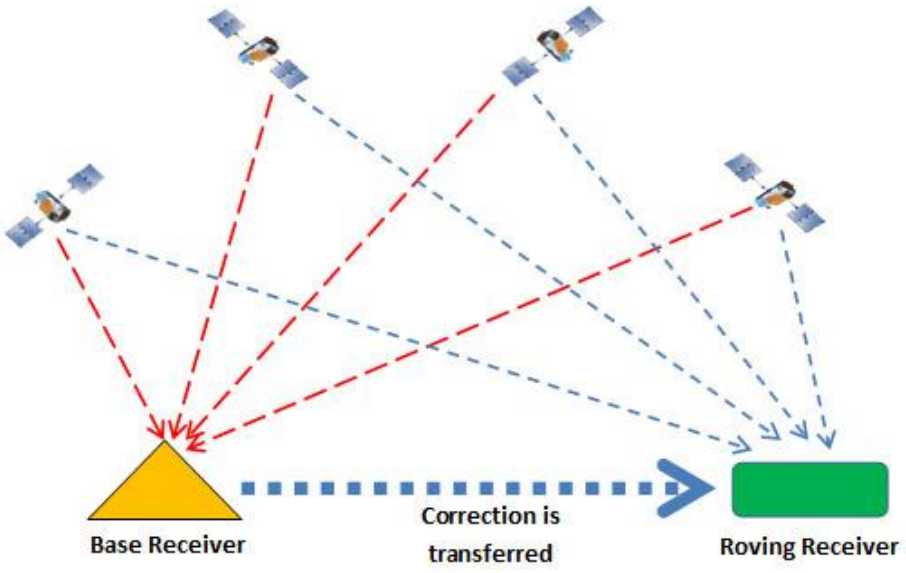


Figura 2.19 GPS de tipo diferencial.

3. Herramientas

Para el desarrollo de este proyecto se han utilizado distintas herramientas informáticas para implementar los programas que permiten conectar con los sensores. La programación se ha llevado a cabo dentro del entorno ROS (Robot Operating System). Los programas están escritos en C++, además se han usado las bibliotecas OpenCV para la detección de peatones. En este capítulo, se empezarán introduciendo las características generales que puede tener cualquier software en general, para pasar después a ver OpenCV y ROS en particular. Finalmente se verán otras arquitecturas que se utilizan en el ámbito de la robótica, campo para el que fue desarrollado el entorno ROS.

3.1 Introducción

La arquitectura de Software es el diseño de más alto nivel de la estructura de un sistema. Tambien se la conoce como arquitectura lógica, y consiste en un conjunto de patrones y abstracciones coherentes que proporcionan un marco al programa.

El tipo de arquitectura se selecciona y se diseña en base a las restricciones y objetivos que se quieren conseguir con el programa a diseñar. Se deben tener en cuenta, tanto los objetivos de tipo funcional a la hora del diseño, como los objetivos de mantenibilidad, flexibilidad, e interacción con otros sistemas de información.

A la hora de crear la arquitectura se deben tener en cuenta además, las restricciones derivadas de las tecnologías disponibles para implementar los programas o los sistemas de información.

La arquitectura de Software define, de una manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos. Además, toda arquitectura debe ser implementable en una arquitectura física, que consiste en determinar que computadora tendrá asignada cada tarea.

Toda arquitectura de Software debe describir diversos aspectos del software. Generalmente, cada uno de estos aspectos se describe mejor utilizando diferentes modelos o vistas de estos. Cada modelo constituye una descripción parcial de una misma arquitectura y es bueno que haya cierto solapamiento entre ellos para que guarden cierta coherencia entre sí.

Los modelos fundamentales en cualquier arquitectura son los siguientes:

-La visión estática: describe qué componentes tiene la arquitectura.

-La visión funcional: describe qué hace cada componente.

-La visión dinámica: describe cómo se comportan los componentes a lo largo del tiempo y como interactúan entre sí.

Estas vistas se pueden expresar en diferentes lenguajes como pueden ser los diagramas de estado, los diagramas de flujo de datos, etc. Cada lenguaje es apropiado para una vista, por ello, desde hace tiempo se está intentando adoptar un único lenguaje con el que poder representar todas las vistas sin cambiar de un lenguaje a otro. El lenguaje Que más éxito ha tenido en esta difícil tarea ha sido UML (Unified Modeling Language). Sin embargo, al usar un único lenguaje se corre el riesgo de no poder expresar ciertas restricciones en una vista de un sistema determinado.

En este sentido de unificar lenguajes y simplificar las cosas, ROS provee de una plataforma que simplifica la programación a la hora de interconectar distintos dispositivos. Además permite usar distintos lenguajes, facilitando así el uso por parte del usuario.

3.2 OpenCV

OpenCV [23] es una biblioteca de funciones de programación principalmente orientada a la visión por computador en tiempo real. Tiene licencia BSD (Berkeley Software Distribution, distribución de software de Berkeley), por lo que es gratis para usos comerciales o de investigación. OpenCV fue originalmente escrita en C, pero ahora tiene una interfaz C++ completa y todos los nuevos desarrollos son en C++. Hay también una interfaz completa de Python de esta biblioteca. Esta disponible para Linux, Mac y Windows.

OpenCV fue diseñada para conseguir eficiencia computacional y para ser usada en aplicaciones de tiempo real. Tiene estructuras básicas de datos para realizar operaciones con matrices y llevar a cabo el procesamiento de imágenes. Permite visualizar datos de una manera sencilla y extraer información de imágenes y videos. Posee funciones de captura y presentación de imágenes.

OpenCV se compone de 5 módulos distintos [24]. El primero es “cv”, el cual contiene las funciones principales de la biblioteca. El segundo es “cvaux”, y contiene las funciones auxiliares o en fase experimental. El tercero es “cxcore”, dentro del cual están contenidas las estructuras de datos y las funciones de soporte para el álgebra lineal. El cuarto es “highgui”, el cual permite al usuario crear y manipular ventanas para mostrar imágenes, así como leer o grabar imágenes o video. El último es “ml” o machine learning, módulo que permite implementar algoritmos de aprendizaje. Se puede ver la estructura en la Fig. 3.1.

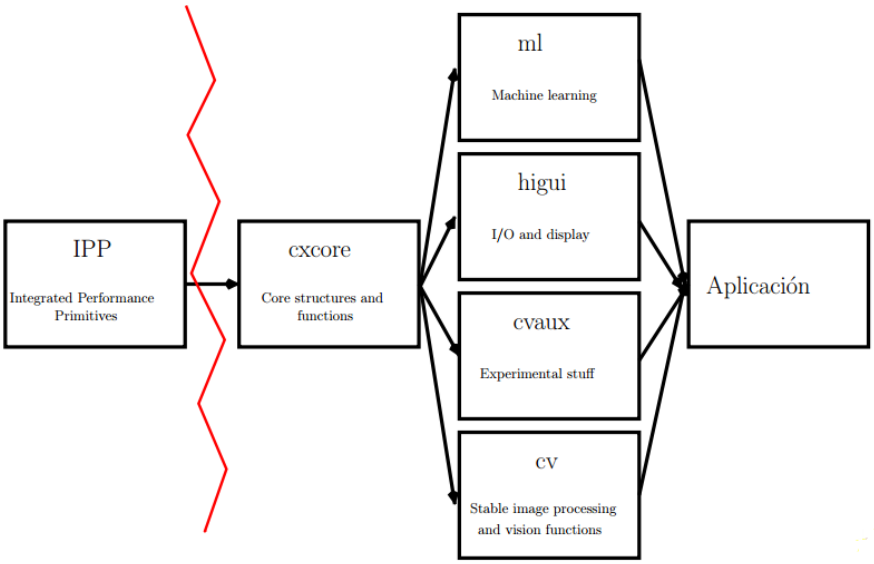


Figura 3.1 Distintos módulos de OpenCV.

Las aplicaciones de las OpenCV abarcan campos muy variados como: HCI (Human Computer Interaction, interacción hombre máquina), identificación de objetos, segmentación y reconocimiento, reconocimiento de caras, reconocimiento de gestos, seguimiento del movimiento, comprensión del movimiento, calibración de sistemas estéreo y multi-cámara y computación de la profundidad, robots móviles y odometria visual. En la Fig. 3.2 se muestran algunas de las herramientas que es posible encontrar entre las muchas funciones de la biblioteca, entre las cuales se halla la que será usada en este proyecto: la detección de peatones.

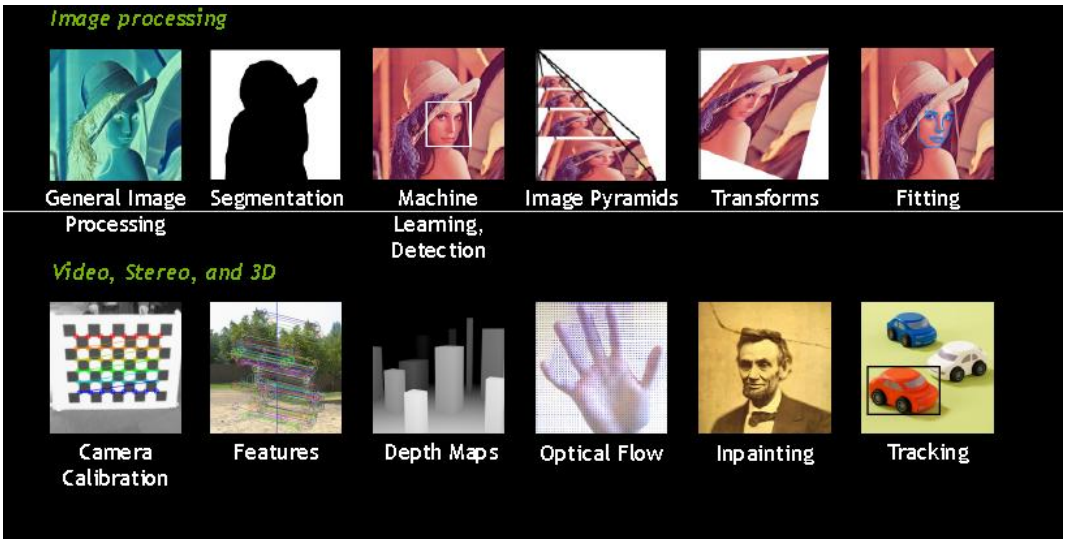


Figura 3.2 Distintas herramientas de OpenCV.

3.3 ROS

El sistema utilizado en este proyecto para implementar los programas ha sido ROS (Robot Operating System) [25]. Este sistema es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un cluster o conjunto heterogéneo. ROS fue desarrollado originalmente en 2007 bajo el nombre de switchyard por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford (STAIR). Desde 2008, el desarrollo ha continuado gracias a Willow Garage, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado.

ROS provee los servicios estándar de un sistema operativo, tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos, que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. La librería está orientada para un sistema UNIX (Ubuntu (Linux) es el sistema soportado aunque también se está adaptando a otros sistemas operativos como Fedora, Mac OS X, Debian o Microsoft Windows considerados como “experimentales” para este sistema).

ROS tiene dos partes básicas: la parte del sistema operativo, ros, como se ha descrito anteriormente, y ros-pkg, una suite de paquetes aportados por la contribución de usuarios (organizados en conjuntos llamados pilas, o en inglés "stacks") que implementan funcionalidades tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc.

El principal objetivo de ROS es conseguir un sistema basado en herramientas, que sea multilingüe, ligero, de código abierto, y "peer-to-peer" o de igual a igual. ROS se construyó basado en estos criterios y a continuación se muestra como esto influye en su arquitectura.

Peer-to-peer:

Un sistema construido usando ROS consiste en un número de procesos, generalmente con un número de patrones diferentes, conectados en tiempo de ejecución siguiendo una topología de igual a igual. Aunque los sistemas basados en un servidor central pueden implementar tambien un diseño de multiproceso y multipatrón, estos pueden dar problemas si se conectan a una red heterogénea.

Por ejemplo, en el tipo de robots para los que fue diseñado ROS, normalmente hay varios ordenadores de a bordo conectados via ethernet. Este segmento de red hace de puente, vía inalámbrica, con las grandes computadoras que están haciendo tareas intensivas de cálculo, tales como reconocimiento de lenguaje o procesamiento en visión artificial. Si ejecutáramos el sistema con un método tradicional de diseño, en cualquiera de las dos subredes de computadoras, de a bordo o de cálculo, podríamos obtener un tráfico innecesario de datos a través de la conexión inalámbrica, porque algunas rutas de mensajes podrían estar contenidas en ambas subredes, la de los ordenadores de a bordo o la de los ordenadores de cálculo. En contraste, con un diseño de conectividad "peer-to-peer", combinado con módulos de software de regulación o "fanout", se evita por completo este problema.

Multilingüe:

Cuando se escribe el código, algunas personas tienen preferencia por escribir en un lenguaje, y otras personas prefieren otro distinto. Estas preferencias son consecuencia de los resultados obtenidos en cuanto al tiempo gastado en la programación, la facilidad de depuración de un programa, la sintaxis usada, la eficiencia del programa cuando esta ejecutándose y otras muchas razones, tanto técnicas como personales. Por todo ello, ROS se diseñó para ser un programa neutro, es decir, ROS soporta varios lenguajes de programación diferentes entre sí, tales como C++, Python, Octave, LISP y será capaz de soportar otros que estan en fase de inclusión.

Aunque es mejor implementar las aplicaciones y funcionalidades en cada lenguaje por separado para tener una visión clara de lo que se hace en cada uno de los lenguajes, en ROS se puede obtener una librería para un tipo de lenguaje, a partir de una librería en otro lenguaje distinto. Esto se hace a través de una conversión, gracias a un lenguaje propio que hace de puente entre los demás.

Para conseguir dar soporte y desarrollar las funcionalidades en todos los lenguajes, ROS utiliza un lenguaje neutral, "Interface Definition Language (IDL)" para definir los mensajes enviados entre módulos. El lenguaje IDL usa archivos de texto cortos para describir los campos de cada mensaje, como por ejemplo el de la Fig. 3.3, en el que se definen dos arrays dentro de la cabecera, uno de tipo Point32 y otro de tipo ChannelFloat32.

```
Header header
Point32[] pts
ChannelFloat32[] chan
```

Figura 3.3 Ejemplo de lenguaje IDL.

El generador de códigos para cada lenguaje soportado, genera entonces unos objetos en el propio lenguaje, y estos son automáticamente publicados por ROS cuando los mensajes son enviados y recibidos. Esto reduce considerablemente el tiempo de programación y los errores cometidos. Las tres lineas anteriores del ejemplo, son automáticamente transformadas en 137 lineas de código en C++, 96 lineas en lenguaje Python, 81 lineas en LISP, y 99 lineas en Octave. Como los mensajes son automáticamente creados desde un archivo de texto muy simple, se pueden crear o añadir nuevos tipos de mensajes de una manera sencilla. A la hora de escribir, ROS tiene cuatrocientos tipos de mensajes distintos los cuales transportan datos que van desde los sensores hasta el mapa de datos del sistema.

El resultado de esto es un diseño a través de un lenguaje neutral, que a traves de distintos tipos de mensajes, y su conversión a los distintos lenguajes, permite tener un esquema en el que se puede elegir el lenguaje de programación que convenga al usuario, o incluso mezclarlos si esto es conveniente.

Basado en herramientas:

En un esfuerzo por gestionar la complejidad de ROS, se optó por usar diferentes núcleos para el procesamiento, es decir, ROS contiene una gran cantidad de pequeñas herramientas que son usadas para construir y ejecutar los diferentes componentes de ROS. Estas herramientas realizan diversas tareas, como el guiado por el árbol de código fuente, obtener y programar los parámetros de configuración, visualizar la topología de conexión de igual a igual, medir la utilización del ancho de banda, trazar gráficamente los datos de mensaje, autogenerar la documentación, y otras muchas. Aunque hay algunas tareas que se implementan como servicios generales, como el reloj global del sistema, se debe intentar colocar todo en módulos separados. La pérdida de eficiencia por usar herramientas como servicios generales no compensa la ganancia de estabilidad en el sistema ni la reducción de la complejidad en la gestión.

Ligero:

La mayoría de los proyectos de software para robots contienen operaciones o algoritmos que podrían ser reutilizados fuera de ese proyecto. Desafortunadamente, por una gran variedad de razones, parte de este código está tan enredado con el "middleware"(software de asistencia de una aplicación para ayudar a esta a comunicarse con otras), que es difícil extraer y reusar la parte del código que interesa en un momento determinado.

Para evitar esto, en ROS se incentiva que todos los algoritmos y operaciones tengan su propia librería independiente sin dependencias en ROS. ROS ejecuta o construye módulos independientes dentro del árbol de código fuente, y el uso de la herramienta CMake hace más fácil esta tarea de simplificar el código. La colocación de prácticamente todo en librerías, creando para ello pequeños ejecutables, revela la funcionalidad de las librerías de ROS. Además, al tener el código dividido en librerías independientes, la prueba y depurado de los programas se vuelve más fácil.

Para facilitar el desarrollo continuo, ROS permite construir las librerías y desarrollar el sistema a partir de fuentes de código externas, como las de repositorios, o las de parches de aplicaciones.

Lenguaje de código abierto:

La totalidad del código fuente de ROS esta disponible para todos. Este es un punto crítico para facilitar el depurado y desarrollo del software en todas las distintas aplicaciones. ROS apuesta por el desarrollo de código abierto, mientras que otros sistemas como Microsoft Robotics Studio tienen un desarrollo llevado a cabo por el mismo propietario, lo que les aporta algunos atributos recomendables, pero que a la vez les hace tener inconvenientes, por ejemplo, a la hora de diseñar o depurar distintas funcionalidades en paralelo.

ROS es distribuido bajo los términos de la licencia BSD (Berkeley Software Distribution), lo que permite el desarrollo de proyectos comerciales y no comerciales. ROS transfiere datos entre módulos usando procesos intermedios de comunicación, y no es necesario que los módulos estén unidos en el mismo ejecutable. Como consecuencia, los sistemas construidos en ROS pueden usar módulos con diferentes tipos de licencia, y los términos de cada licencia solo se aplican a los módulos con ese tipo de licencia específica.

Nomenclatura:

Los términos fundamentales de implementación en ROS son los nodos, los mensajes, los tópicos, y los servicios.

- **Nodos**: Los nodos son procesos que realizan algún tipo de computación. Como ROS esta diseñado para ser modular, un sistema normalmente está compuesto por varios nodos. En este contexto, se puede intercambiar la palabra "nodo" por "módulo de software". Cuando se tienen varios nodos ejecutandose a la vez, es conveniente reproducir las comunicaciones entre estos nodos en forma de gráfico, siendo cada nodo en ejecución un nodo del gráfico y las comunicaciones o uniones entre estos un arco.

- **Mensajes**: Los nodos se comunican entre ellos por medio de mensajes. Estos mensajes pueden estar compuestos de datos de tipo primitivo, como enteros o flotantes, de matrices de datos primitivos, o de constantes. Además, los mensajes pueden estar formados por otros mensajes o matrices de otros mensajes.

Un nodo envía un mensaje para que sea publicado en un tópico determinado, el cual es simplemente una cadena. Un nodo que está interesado en un cierto tipo de información deberá suscribirse al tópico adecuado. Para un tópico puede haber varios publicadores y suscriptores, además, un nodo puede publicar o suscribirse a varios tópicos. En general, los publicantes o suscriptore desconocen la existencia de otros.

El tipo de comunicación más simple entre nodos es a través de tuberias o con relación entre nodos de uno a uno como se puede ver en la Fig. 3.4, en la que se establece una comunicación mediante una relación de uno a uno en el proceso que transcurre entre la recepción de un sonido por un micrófono y la emisión de este sonido mediante un altavoz.

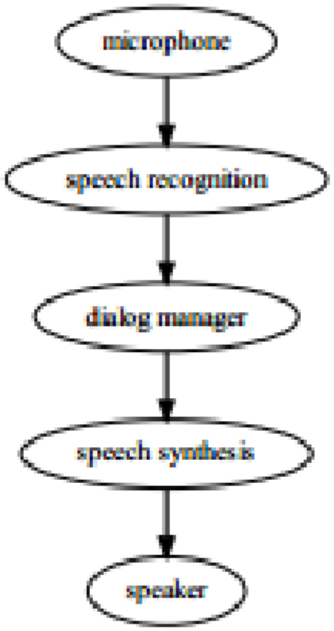


Figura 3.4 Representación gráfica de la comunicación a través de tuberias.

Sin embargo, los gráficos son normalmente mucho más complejos, conteniendo a menudo relaciones de uno a muchos o muchos a uno, como la que se muestra en la Fig. 3.5.

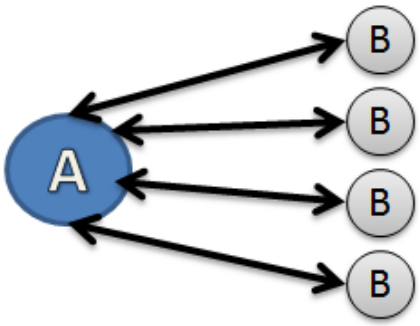


Figura 3.5 Representación gráfica de relación uno a muchos.

3.4 Otras arquitecturas

3.4.1 Middleware vs framework

Middleware

Middleware es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos [26]. Es el software que proporciona un enlace entre aplicaciones de software independientes. A veces se llama middleware a la vía que conecta dos aplicaciones y pasa los datos entre ellas. Los middleware permiten que los datos contenidos en una base de datos puedan ser accedidos a través de otra, ahorrando tiempo a los programadores. Esto simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en los sistemas distribuidos. De esta forma, se provee una solución que mejora la calidad de servicio, seguridad, envío de mensajes, directorio de servicio, etc.

Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware se abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API (Application Programming Interface) para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipos de servicios de middleware. Por lo general el middleware del lado cliente está implementado por el Sistema Operativo, el cual posee las bibliotecas que ejecutan todas las funcionalidades para la comunicación a través de la red.

Framework

En el desarrollo de software, un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio, y provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza a las aplicaciones del dominio.

Los frameworks tienen como objetivo ofrecer una funcionalidad definida, auto contenida, siendo construidos usando patrones de diseño, y su característica principal es su alta cohesión y bajo acoplamiento. Para acceder a esa funcionalidad, se construyen piezas u objetos, llamados objetos calientes, que vinculan las necesidades del sistema con la funcionalidad que este presta. Esta funcionalidad, esta constituida por objetos llamados fríos, que sufren pocos o ningún cambio en la vida del framework, permitiendo la portabilidad entre distintos sistemas. Frameworks conocidos que se pueden mencionar por ejemplo son Spring Framework, o Hibernate (para Java Netbeans), donde lo esencial para ser denominados frameworks, es estar constituidos por objetos cuasi estáticos con funcionalidad definida a nivel de grupo de objetos y no como parte constitutiva de estos, por ejemplo en sus métodos, en cuyo caso se habla de un API (Application Programming Interface) o librería.

3.4.2 RobotStudio

Ya que ROS es ampliamente utilizado para controlar robots, a continuación se ven las principales características de otro programa que se encarga de controlar robots, RobotStudio, aunque en este caso, RobotStudio está enfocado a robots industriales y no a robots experimentales [27].

La programación offline es la mejor forma de aumentar la rentabilidad de la inversión en sistemas de robots. La simulación de ABB y el software de programación offline RobotStudio, permiten programar los robots en un PC sin necesidad de parar la producción. También se pueden preparar los programas de los robots anticipadamente, lo que implica un aumento de la productividad.

RobotStudio aporta herramientas que aumentan la rentabilidad del sistema de robots, pues permite realizar tareas tales como formación, programación y

optimización de programas sin alterar la producción. Esto añade muchas ventajas, entre ellas:

- Reducción de riesgos.

- Arranques más rápidos.

- Menor tiempo para modificaciones.

- Aumento de la productividad.

RobotStudio se basa en el controlador virtual de ABB, una copia exacta del software real que emplean los robots en la producción. Ello permite ejecutar simulaciones muy realistas, utilizando programas de robots reales y archivos de configuración idénticos a los que se emplean en la fábrica.

Importación de CAD: RobotStudio permite importar fácilmente datos de los principales formatos de CAD, como IGES, STEP, VRML, VDAFS, ACIS y CATIA. Al trabajar con esta información de alta precisión, el programador del robot puede generar programas de robot más exactos, lo que da lugar a una mayor calidad del producto.

AutoPath: Ésta es una de las funciones de RobotStudio que más tiempo ahorra. Al utilizar un modelo de CAD de la pieza que se desea programar, es posible generar automáticamente y en cuestión de minutos las posiciones de robot que se necesitan para seguir la curva, algo que de lo contrario requeriría horas o incluso días.

AutoReach: AutoReach analiza automáticamente las posibilidades de alcance y es una función muy práctica que permite mover simplemente el robot o la pieza de trabajo hasta que todas las posiciones puedan alcanzarse. Esto permite verificar y optimizar la disposición de la célula de trabajo en cuestión de minutos. Se puede ver un ejemplo en la Fig. 3.6 en la que los ejes amarillos muestran las distintas posiciones que adoptará el robot.

Optimización de trayectorias: RobotStudio puede detectar automáticamente los programas que contengan movimientos demasiado cercanos a las singularidades (puntos conflictivos) y advertir automáticamente de estos problemas, para que sea posible tomar medidas para evitar estas situaciones.

El monitor de simulación es una herramienta visual que permite optimizar los movimientos del robot.

Las líneas de color rojo indican qué objetivos pueden mejorarse para que el robot se mueva de la forma más efectiva. Es posible optimizar la velocidad del TCP, la aceleración, la singularidad o los ejes para mejorar los tiempos de ciclo.

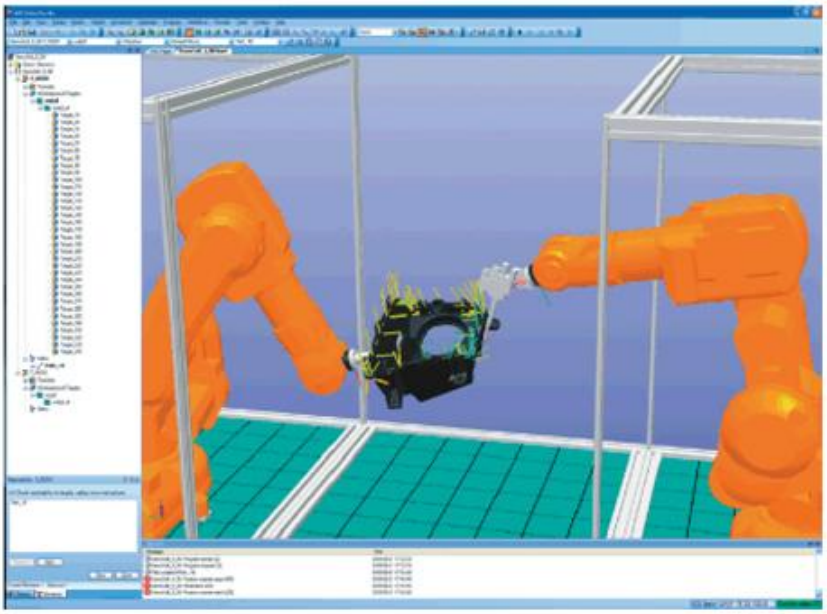


Figura 3.6 Ejemplo de la funcionalidad AutoReach

Detección de colisiones: La detección de colisiones previene los costosos daños en los equipos. Al seleccionar los objetos afectados, RobotStudio controlará e indicará automáticamente si colisionarán cuando se ejecute el programa. Un ejemplo de ello se ve en la Fig. 3.7.

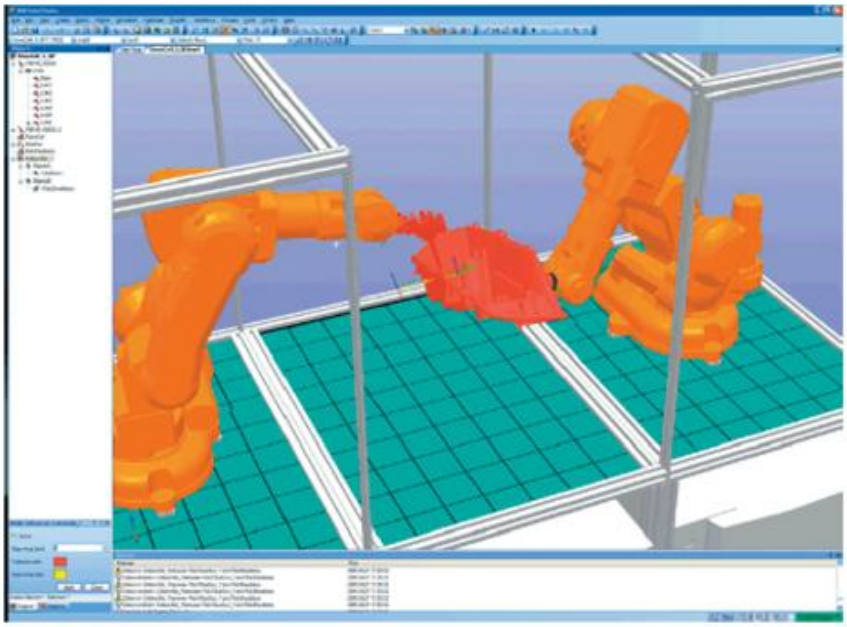


Figura 3.7 Ejemplo de la detección de colisiones

FlexPendant virtual: Se trata de una representación gráfica de la unidad de programación táctil real, pero controlada por VirtualRobot. Básicamente, todo lo que puede hacerse en la unidad de programación táctil real puede hacerse en la Unidad de Programación Táctil, lo que hace de esta herramienta el medio ideal para labores de enseñanza y prácticas.

Carga y descarga reales: Todo el programa de robot puede transferirse al sistema real sin necesidad de ninguna conversión. Se trata de una característica exclusiva gracias a la tecnología VirtualRobot, sólo ofrecida por ABB.

MultiMove: Es posible ejecutar varios robots virtuales a la vez y por ello se incorpora la compatibilidad entre robots con MultiMove, la nueva tecnología de IRC5 para la ejecución de varios robots desde un mismo controlador. En la Fig. 3.8 se muestrán dos robots trabajando simultáneamente con la ayuda de esta tecnología.

Figura 3.8 Ejemplo de la funcionalidad MultiMove

3.4.3 VxWorks

VxWorks es un sistema operativo de tiempo real, vendido y fabricado por Wind River Systems [28]. Como la mayoria de los sistemas operativos en tiempo real, VxWorks incluye kernel multitarea con planificador preemptive (los procesos pueden tomar la CPU arbitrariamente), respuesta rápida a las interrupciones, comunicación entre procesos, sincronización y sistema de archivos.

Las características distintivas son:

- la compatibilidad POSIX.

- el tratamiento de memoria.

- las características de multiprocesador.

- una shell de interfaz de usuario.

- monitor de rendimiento y depuración de código fuente y simbólico.

VxWorks se utiliza generalmente en sistemas embebidos. Sus ventajas son que no utiliza mucha memoria, cualquier evento en el soporte físico puede hacer que se ejecute una tarea, es multiarquitectura (el código puede ser portado a cualquier CPU), es multitarea con herramientas de comunicación entre procesos, como semáforos señales y sucesos, usa una planificación preferente basada en prioridades y reduce los intervalos en los que están inhabilitadas las interrupciones.

Sus desventajas son que la cantidad de tiempo necesario para iniciar la gestión de una interrupción y comenzar la ejecución de su rutina de tratamiento, el efecto del tratamiento de interrupciones (el servicio se retrasará si una interrupción puede ser interrumpida por otra) y la necesidad de estabilidad, ya que un fallo en el sistema puede ser catastrófico.

3.4.4 Microsoft Robotics Studio

Microsoft Robotics Studio es un entorno de desarrollo orientado al desarrollo de aplicaciones en robótica. Una de las ventajas de MRDS es proporcionar una plataforma de desarrollo de robótica con soporte de concurrencia en tiempo real orientado a servicios y una plataforma escalable y extensible.

Los principales componentes de Robotics Studio son:

- Un Lenguaje de Programación Visual (VPL), que permite la creación intuitiva de aplicaciones para robots.
- Un entorno de simulación Visual 3D basado en el motor de simulación física AGEIA.
- Soporte en tiempo de ejecución (Runtime) que gestiona la entrada/salida asíncrona, la concurrencia y la distribución de servicios.

MRDS está diseñado para ser una plataforma de desarrollo genérica que pueda emplearse con gran diversidad de fabricantes de robots, existen gran variedad de servicios que operan directamente con los modelos de robots más populares. Una aplicación en Robotics Studio es en esencia una coordinación de diversos servicios distribuidos y asíncronos. Por ejemplo, el sensor de contacto se representa por un

servicio que ofrece una entrada de información, un motor o actuador tendrá otro servicio asociado que representa la respuesta física del robot, finalmente, un servicio controlador se encargaría de interpretar la información obtenida del sensor y mandaría los comandos apropiados al actuador, para que este realize una determinada operación.

La coordinación se produce en la comunicación asíncrona entre todos estos servicios. Por ejemplo, si el servicio de parachoques detecta un impacto, éste envía un mensaje al servicio controlador, que a su vez decidirá qué mensaje enviar al servicio de control de ruedas para realizar la maniobra oportuna. Este escenario se complica cuando el número de sensores y de actuadores aumenta, sin embargo, el servicio coordinador (o servicios coordinadores) debe manejar mucha información en tiempo real y aplicar complejas políticas de control.

El soporte de tiempo de ejecución o Runtime de MRDS consta de dos componentes principales que hacen posible la construcción, supervisión, despliegue y funcionamiento de un gran rango de aplicaciones. Estos dos componentes son el CCR (Concurrency and Coordination Runtime) y el DSS (Decentralized Software Services).

El CCR permite la coordinación concurrente y asíncrona del flujo de ejecución abstrayendo al programador del uso de hilos, semáforos y otras técnicas de más bajo nivel para el aseguramiento de la exclusión mútua o la prevención del interbloqueo. Además plantea un modelo de programación asíncrona que facilita y optimiza la explotación de un entorno de ejecución paralelo o multihilo. Cabe destacar que el CCR es un componente DLL que se ejecuta en el entorno .NET y accesible desde cualquiera de los lenguajes de programación disponibles en .NET.

El DSS combina la arquitectura tradicional Web (HTTP) con elementos de las nuevas arquitecturas orientadas a servicios Web (SOAP). La arquitectura resultante está completamente basada en servicios que se coordinan entre sí para crear aplicaciones distribuidas. Por lo tanto, desde este punto de vista, una aplicación desarrollada en Robotics Studio es un conjunto de servicios que se coordinan entre sí. El principal objetivo es promover la simplicidad, transparencia y la interoperabilidad. Las composiciones de servicios se pueden usar sin importar si estos servicios están funcionando dentro del mismo nodo o a través de la red. El resultado es una plataforma flexible capaz de soportar un amplio rango de aplicaciones. El DSS utiliza los protocolos HTTP y DSSP. DSSP es un protocolo propio que ofrece DSS y se encarga de la mensajería entre servicios. Los servicios mantienen un estado durante el periodo de vida de la aplicación y se ejecutan en nodos DSS, que son los encargados de posibilitar la comunicación entre todos los servicios.

4. Desarrollo del proyecto

En este capítulo se resume el desarrollo del proyecto. En una primera parte se muestra un pequeño manual para entender un poco mejor el funcionamiento de ROS. Se explican las cosas más importantes para poder entender la estructura y comunicación de los programas usados en el proyecto.

En una segunda parte, se explica el desarrollo del proyecto y los programas específicos usados en este.

4.1 Primeros pasos con ROS

Esta primera parte pretende servir para entender un poco mejor el funcionamiento de ROS. Si se quiere tener una información más detallada se puede acudir a la zona de tutoriales de la página oficial de ROS [29].

ROS es un meta-sistema operativo para robots, es decir, una librería de código que debe ser instalada en un sistema operativo y que será invocada desde un programa ejecutable. Entre sus ventajas destaca que se abstrae del hardware, controla los dispositivos a bajo nivel, implementa las funcionalidades más comunes en el manejo de robots, intercambia mensajes entre procesos y es capaz de administrar paquetes de código. También proporciona librerías y herramientas para obtener, construir, escribir y ejecutar código a través de múltiples ordenadores.

ROS engloba una estructura o red del tipo p2p (peer to peer o puesto a puesto) donde los procesos se comunican entre ellos usando la infraestructura de comunicaciones de ROS. Esta comunicación es síncrona cuando se utilizan servicios (comunicación tipo petición/respuesta). Es asíncrona cuando se usan topics o temas; un topic es un bus por donde circulan mensajes intercambiados por nodos, que son pequeños programas que ejecutan tareas.

Para programar elementos con ROS pueden usarse 3 lenguajes: c++, python y Lisp (LISt Processing es un lenguaje con mas de 50 años de historia que en la actualidad se usa principalmente para el desarrollo de aplicaciones y nuevas bibliotecas portátiles).

4.1.1 Sistemas de ficheros ROS

En cuanto a sus archivos, ROS se compone de:

- Paquetes (packages): Un paquete puede contener nodos (procesos ejecutables), librerías, conjuntos de datos (datasets), archivos de configuración, etc.

- Manifiestos (manifests): son archivos con extensión .xml que proporcionan información (metadatos), sobre todo de dependencias del paquete.

- Pilas (stacks): Son colecciones de paquetes.

- Manifiestos de Pila: Igual que los de paquetes.

- Tipos de Mensajes: Archivos con extensión .msg. Las descripciones de los mensajes están almacenadas en my_package / msg / MyMessageType.msg y describen la estructura de los mensajes enviados en ROS.

- Tipos de Servicios: Archivos con extensión .srv. Las descripciones de los servicios están almacenadas en my_package / srv / MyServiceType.srv y definen las peticiones y respuestas de las estructuras de datos para los servicios en ROS.

4.1.2 Elementos de computación de ROS

- Nodos: Los nodos son procesos o programas independientes ejecutables. ROS está diseñado para ser modular. Por ejemplo, un robot tendrá un nodo para controlar el láser, otro nodo para controlar los motores de desplazamiento, otro nodo para calcular el camino correcto, etc… Un nodo se programa utilizando una librería cliente ROS, como por ejemplo “roscpp” o “rospy”, si se va a programar en c++ o pyhton respectivamente.

- Maestro (master): asigna nombres a todos los elementos que conforman el modelo del robot. Sin él, los nodos no se “encontrarían” ni podrían intercambiar mensajes entre ellos.

- Servidor de Parámetros: Forma parte del Maestro y permite almacenar datos por clave en una localización centralizada.

- Mensajes: Son elementos que utilizan los nodos para comunicarse entre ellos. Los mensajes son simples estructuras de datos compuestas por campos tipificados.

- Temas (topics): Son mensajes transportados mediante semántica de publicación/subscripción. Un nodo publica un mensaje en un topic dado; El topic es un nombre usado para identificar el contenido del mensaje; Un nodo que esté interesado en un cierto tipo de datos, se subscribirá al topic apropiado. Puede haber múltiples publicadores y subscriptores para un único topic y un único nodo puede publicar y/o subscribirse a múltiples topics. La idea es desconectar los proveedores de información de los consumidores para conseguir una programación modular. En la Fig. 4.1. se puede ver un esquema de la comunicación entre nodos usando topics.

- Servicios: Los servicios son otro elemento mediante el cual se pueden comunicar los nodos. Un nodo cliente envía una solicitud a un servicio determinado creado por

otro nodo, y cuando el segundo nodo recibe esta solicitud, envía una respuesta al cliente. Se puede entender como el modelo petición-respuesta que se usa en internet. Por ejemplo, si se tiene un nodo que ofrece un servicio que consiste en sumar dos numeros que se le pasan y devolver el resultado, se puede crear un nodo cliente que envíe un par de números a este nodo, de forma que, al recibir la petición suma los dos números y devuelve la respuesta al cliente, que en este caso es la suma de los números.

- Bolsas (bags): La bolsa es un mecanismo para guardar mensajes y reproducirlos posteriormente. Es una herramienta de mucha utilidad, por ejemplo para guardar datos recogidos de sensores y usar estos datos para trabajar sin necesidad de los sensores.

Los nombres son muy importantes en ROS. Todos los elementos computacionales en ROS tienen nombre: los nodos, los topics, los servicios y los parámetros. Todas las librerías cliente de ROS soportan el remapeo de nombres por línea de comandos, lo que permite que un programa ya compilado pueda ser reconfigurado en tiempo real para operar en una topología computacional diferente.

Por ejemplo, para controlar un láser, se puede iniciar el nodo-driver láser, el cual dialoga con el dispositivo láser y publica mensajes del tipo “LaserScan” en el topic “scan”. Para procesar esos datos, se debe programar un nodo usando filtros láser que se suscriben a los mensajes del topic “scan”. Tras la subscripción, el filtro podría automáticamente iniciarse recibiendo mensajes desde el láser.

Los dos extremos o nodos del ejemplo no estan conectados entre sí. El nodo láser publica datos del láser sin saber quién está subscrito. El nodo filtro está subscrito al sensor sin saber qué nodo publica sus datos. Los dos nodos pueden ser iniciados, finalizados y reiniciados, en cualquier orden, sin que se produzcan errores.

Si más tarde se añade otro láser al robot, se necesitará reconfigurar el sistema. Pero gracias a la arquitectura ROS, todo lo que se necesita es remapear los nombres de los láseres que son usados. Cuando se inicia el primer nodo láser, se le puede indicar que remapee su nombre de “scan” a “base_scan” y hacer lo mismo con el nodo filtro. Ahora esos nodos se comunicarán usando el topic “base_scan” sin escuchar ya mensajes del topic “scan”. Hecho esto se puede iniciar un nuevo nodo láser, que usará el topic “scan”, y por tanto será totalmente independiente del otro nodo láser.

De esta manera se consiguen controlar varios láseres en tiempo real de una manera sencilla.

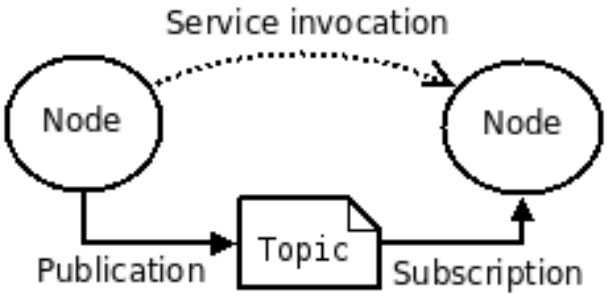


Figura 4.1 Representación esquemática del funcionamiento de los nodos.

4.1.3 Instalación de ROS

Para empezar, se tiene que configurar el sistema para que acepte el software de paquetes que proceden de packages.ros.org. Para ello se debe ejecutar el siguiente comando:

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'

Si la versión de Ubuntu no es 12.04 (Precise) se sustituye el nombre "precise" por el nombre de la versión de Ubuntu que se tenga, por ejemplo "quantal".

A continuación se deben configurar los comandos del sistema con el siguiente comando:

wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

Ahora hay que asegurarse de que el índice de paquetes del sistema está actualizado. Para ello se usa el siguiente comando:

sudo apt-get update

Ahora se puede elegir entre diferentes paquetes y herramientas de ROS para instalar. Si se quiere la instalación completa, que incluye el sistema ROS y librerías de robots genéricos, de simulaciones 2D y 3D, de navegación y de sistemas de percepción, se debe ejecutar el siguiente comando:

sudo apt-get install ros-hydro-desktop-full

En este caso se pretende instalar la versión hydro de ROS. Si se quisiera instalar otra, se debe sustituir la palabra "hydro" por el nombre de la versión que se quiera, por ejemplo "fuerte".

Antes de empezar a usar ROS se debe instalar rosdep, que es una herramienta que facilita la instalación y compilación de paquetes de fuentes externas y además es necesario para algunas aplicaciones de ROS. Para ello se ejecutan los siguientes comandos:

sudo rosdep init

rosdep update

Es conveniente configurar el entorno para que las nuevas aplicaciones sean reconocidas directamente por ROS. Para ello se introducen los comandos:

echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc

source ~/.bashrc

Si se tienen varias versiones de ROS instaladas, es posible que se necesite utilizar una u otra versión dependiendo del caso. Para ello, se escribe la versión de ROS que se quiera usar en el siguiente comando y se ejecuta:

source /opt/ros/hydro/setup.bash

En este caso se esta configurando el terminal para que funcione utilizando la versión hydro. Si se tienen varios terminales abiertos, se puede configurar la versión a usar en cada terminal por separado.

4.1.4 Trabajando con ROS

En primer lugar hay que asegurarse de que ROS está correctamente instalado y de que está definido de forma correcta. Se ejecuta en un terminal la siguiente instrucción:

export | grep ROS

Como resultado, en la pantalla del terminal deben aparecer una serie de definiciones como el path de ROS, el host_name, el local host, etc…

Ahora se debe crear un entorno de trabajo para trabajar con ROS. Este proporcionará una carpeta donde crear paquetes y pilas propios.

Los siguientes comandos crean un espacio de trabajo dentro de una carpeta a la que se define con el nombre hydro_workspace, y que puede acceder a los paquetes ubicados en /opt/ros/hydro, que es la carpeta principal en la que se encuentran los archivos:

rosws init ~/hydro_workspace /opt/ros/fuerte

mkdir ~/hydro_workspace/sandbox

rosws set ~/hydro_workspace/sandbox

El entorno o espacio de trabajo (Workspace) está definido en la variable de entorno ROS_PACKAGE_PAT. Se puede ver su valor introduciendo el comando:

echo \$ROS_PACKAGE_PATH

En la distribución hydro el resultado devuelto por pantalla será:

/opt/ros/hydro/stacks

La estructura de las carpetas en el disco duro parte de la raíz del entorno de trabajo (/opt/ros/hydro/stacks). Todas las carpetas que cuelguen de ahí, son pilas o stacks. Dentro de cada pila puede haber varios paquetes (varias subcarpetas) o un solo paquete. En el caso de que la pila contenga un único paquete se puede ver que el directorio tiene carpetas como bin, config, launch o src. Además, contiene archivos como manifest.xml, etc.

Para poder usar ROS hay que iniciar su estructura mediante la instrucción roscore. Si se intenta ejecutar roscore en un terminal cuando ya se ha ejecutado en otro terminal, el sistema mostrará un aviso (en rojo) indicando que ya se encuentra en ejecución, como se observa en la Fig. 4.2.

Figura 4.2 Fallo al intentar ejecutar roscore de nuevo.

ENTENDIENDO LOS NODOS

Ya se tiene roscore ejecutándose en un terminal. Ahora se debe abrir otro terminal (sin cerrar el de roscore) para probar el comando rosnode. Para ello, en el nuevo terminal hay que ejecutar:

rosnode list

Con ello se muestran en pantalla todos los nodos en ejecución en ese momento como se ve en la Fig. 4.3.

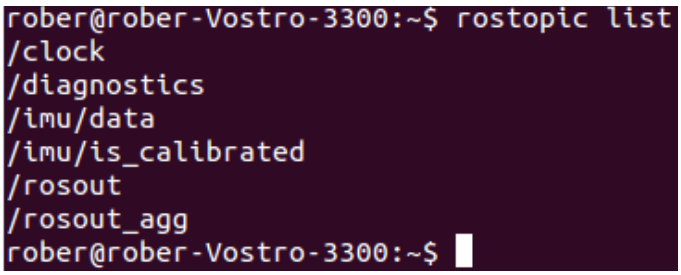


Figura 4.3 Nodos en ejecución.

Si queremosse quiere tener más información sobre un paquete concreto se escribe el comando “rosnode info /nombre_paquete”. Por ejemplo, para el paquete /imu/data se escribe:

rosnode info /imu/data

Con esto se podrán ver sus publicaciones, subscripciones, servicios, etc…

Para ejecutar un nodo se utiliza el comando “rosrun [paquete] [nodo]”. Se puede ejecutar un nodo sin conocer la ruta de su paquete mediante el comando rosrun. Por ejemplo en el caso del sensor inercial utilizado en el proyecto, se introduce directamente el nodo, que en este caso, al encontrarse el sensor conectado al puerto USB número 0 es:

rosrun microstrain_3dmgx2_imu imu_node _port:=/dev/ttyUSB0

ENTENDIENDO LOS TOPICS

Un topic es un medio de comunicación entre dos o más nodos, donde uno o más nodos son los que envían información al topic, y uno o más nodos reciben esta información. Para enviar o recibir la información de un topic, los nodos deben estar suscritos a este topic.

ROS tiene una herramienta muy útil para ver gráficamente los nodos y topics en ejecución y sus relaciones, RXGraph. Para abrirla se ejecutael comando:

rxgraph

Por ejemplo, para los tutoriales de ROS, esta herramienta muestra el esquema mostrado en la Fig. 4.4.

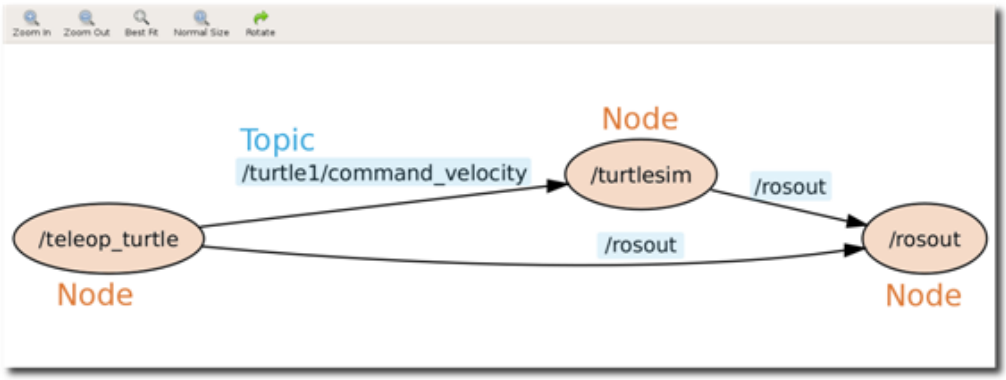


Figura 4.4 Programa rxgraph.

Este gráfico indica que el nodo /teleop_turtle se suscribe al topic /turtle1 /command_velocity para enviar datos, y el nodo /turtlesim se suscribe a este mismo topic para recibir esos datos. A su vez, los dos nodos estan conectados con el proceso principal /rosout, que es el que se ejecuta al lanzar rosnode.

Un servicio es otra forma en la que un nodo puede comunicarse con el resto. Los servicios permiten a los nodos enviar una petición y recibir una respuesta.

Con el comando rosservice se pueden ver los distintos servicios que se pueden utilizar sobre un nodo mediante el comando “list” e interactuar con el nodo mediante el comando “call”.

El comando “rosparam” permite almacenar y manipular datos en el servidor de parámetros ROS. El servidor de parámetros puede almacenar enteros, flotantes, booleanos, diccionarios y listas. Un diccionario se representa como {a: b, c: d}, es decir un array de parejas de datos, en las que el primer dato es el nombre y el segundo es el valor asociado a ese nombre. Una lista se representa como [1, 2, 3], que corresponde con unn array simple de datos. El servidor de parámetros es un diccionario compartido accesible por todos los nodos ROS. Los nodos usan este servidor para almacenar y recuperar datos en tiempo real. Estos datos suelen utilizarse como parámetros de configuración. En la Fig. 4.5 se puede ver como se modifica el parámetro red de background. Primero se modifica mediante “set”, y luego se muestra mediante “get” para ver el resultado.

```
$ rosparam set background_r 150
```

```
$ rosparam get /
```

```
background_b: 255
background_g: 86
background_r: 150
roslaunch:
  uris: {'aqy:51932': 'http://aqy:51932/'}
run_id: e07ea71e-98df-11de-8875-001b21201aa8
```

Figura 4.5 Modificación de un parámetro con rosparam.

CREANDO UN PAQUETE ROS

Un paquete ROS se compone de diferentes archivos: archivos de manifiesto .xml, CMakeLists.txt, Makefiles, mainpage.dox, … Existe un comando ROS que automatiza

la tarea de compilación de un paquete: “roscreate-pkg”. Para crear un paquete en el directorio actual se utiliza el siguiente comando:

roscreate-pkg [nombre del paquete]

Si dicho paquete tiene dependencias de otros paquetes, entonces se utiliza:

roscreate-pkg [nombre del paquete] [dependencia1] [dependencia2] ...

Una vez creado el paquete, hay que asegurarse de que este sea visible para ROS. A menudo es útil ejecutar “rospack profile” para que se reflejen los cambios de directorios en el PATH del paquete. Si se quiere buscar un paquete determinado, se puede hacer utilizando “rospack find [nombre del paquete]”.

CREANDO ARCHIVOS MSG Y ARCHIVOS SRV

Estos archivos de texto se utilizan para describir los elementos y parámetros de los mensajes y servicios. Si por ejemplo, se programa un nodo servicio con 2 parámetros de entrada y uno de salida, estos parámetros hay que definirlos en un archivo .srv asociado a dicho servicio.

Los archivos msg son simples archivos de texto que describen los campos de un mensaje ROS. Estos archivos son utilizados para generar código fuente de los mensajes en diferentes lenguajes durante el proceso de compilación.

Los archivos .srv describen un servicio. Se componen de dos partes separadas por guiones: una petición y una respuesta.

Los archivos .msg se almacenan en el directorio /msg dentro del paquete, y los archivos .srv dentro del directorio /srv.

Un ejemplo de archivo msg es, por ejemplo, el de la Fig. 4.6. en el que se define una cadena de caracteres con un identificador child_frame_id y dos datos del tipo especificado definidos en el archivo geometry_msgs.

```
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Figura 4.6 Ejemplo de msg.

Los archivos .srv son similares a los msg, pero la diferencia es que los de tipo srv. tienen dos partes: una petición al principio y una respuesta tras los tres guiones. Un

ejemplo de archivo .srv es el de la Fig. 4.7, en el que se definen dos datos de entrada y uno de salida, todos ellos de tipo entero largo.

```
int64 A
int64 B
---
int64 Sum
```

Figura 4.7 Ejemplo de srv.

Para crear un mensaje se crea un fichero .msg en la carpeta /msg del paquete con el contenido deseado.

Para asegurar que se generará el mensaje cuando se compile el paquete, hay que abrir el archivo CMakeLists.txt (contenido en el paquete) en un editor y quitar el símbolo de comentario (#) de la siguiente línea:

 # rosbuild_genmsg()

Para crear un servicio se genera previamente un archivo dentro de la carpeta /srv con el contenido de los parámetros a utilizar por dicho servicio.

Se debe quitar el símbolo (#) de la siguiente linea en el mismo archivo CMakeLists.txt del caso anterior:

 # rosbuild_gensrv()

Finalmente, después de crear un archivo .msg y/o un archivo .srv, hay que volver a compilar el paquete por medio del comando:

rosmake [nombre del paquete]

ESCRIBIENDO UN NODO PUBLICADOR Y UNO SUSCRIPTOR EN C++

Lo primero que hay que hacer es desplazarse con roscd al directorio del paquete con el que se quiere trabajar, por ejemplo, para los tutoriales de ROS:

roscd beginner_tutorials

Ahora, dentro de la carpeta /src, se crea un archivo (para este ejemplo, talker.cpp) con el siguiente contenido:

```
#include "ros/ros.h"                //Obligatorio ponerlo para trabajar con ROS
#include "std_msgs/String.h"        //El tipo String de ROS

#include <sstream>
```

```
* Este tutorial muestra cómo enviar mensajes al sistema ROS

int main(int argc, char **argv)
{

  * La función ros::init() inicializa el nodo “talker” en el sistema ROS.  Se le pasan los argumentos para la línea de comandos y un tercer argumento con el nombre del nodo.

 ros::init(argc, argv, "talker");

  * NodeHandle es el principal punto de acceso para comunicarse con el sistema ROS

 ros::NodeHandle n;

  * La función advertise() sirve para decirle a ROS que se quiere publicar en un topic determinado.
  * (en este caso el topic se llama “chatter” y se esta indicando que el mensaje será de tipo String)
  * Con esto se hace una llamada al nodo maestro de ROS, el cual lleva un registro de qué nodos están
  * publicando y qué nodos están subscritos. advertise() devuelve un objeto publicador que permite
  * publicar mensajes en el topic mediante una llamada a la función publish().
  * El segundo parámetro de advertise() es el tamaño de la cola del mensaje usada para publicar mensajes.
  * Si los mensajes son publicados más deprisa de lo que puedan enviarse, se guardarán en la cola.

 ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

 ros::Rate loop_rate(10);  //Define la frecuencia en hercios para temporizar (10 veces por segundo).

  * Contador con el número de mensajes enviados. Se usa para crear un único string por cada mensaje.

 int count = 0;
 while (ros::ok()) //Mientras el sistema ROS esté operativo
 {

   * Objeto mensaje donde se cargan los datos para después publicarlos.

  std_msgs::String msg;

  std::stringstream ss;
  ss << "hello world " << count;
  msg.data = ss.str(); //data es un miembro del tipo String. String es un archivo .msg

  ROS_INFO("%s", msg.data.c_str()); //este es el equivalente ROS a printf y cout

   * Con la función publish() enviamos los mensajes. El parámetro es el objeto mensaje
   * El tipo del objeto debe coincidir con el que se puso en advertise()

  chatter_pub.publish(msg);

   * Aquí no es necesaria, pero spinOnce() se pone para atender a las funciones callback.
   * Si se hubiese añadido una subscripción en esta aplicación, no funcionaría sin esta llamada.

  ros::spinOnce();

  loop_rate.sleep(); //Aplica el temporizador que se definió antes. Publica 10 veces/seg
  ++count;
 }


 return 0;
}
```

Para crear este nodo en el paquete, se debe editar el archivo “CMakeLists.txt” del directorio del paquete y añadir al final la siguiente línea:

Rosbuild_add_executable(talker src/talker.cpp)

Esto creará un ejecutable “talker” del archivo talker.cpp, previamente escrito, que estará ubicado en la carpeta src, la cual por defecto, a su vez se ubicará en la carpeta /bin.

Ahora que está creado el publicador, hay que crear el nodo suscriptor. Se crea el archivo “listener.cpp” dentro de la carpeta /src.

```
#include "ros/ros.h"                    //Obligatorio ponerlo para trabajar con ROS
#include "std_msgs/String.h"        //El tipo String de ROS

* Este tutorial muestra un simple nodo receptor de mensajes.

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
 ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{

  * init() inicializa el nodo “listener” en el sistema ROS

 ros::init(argc, argv, "listener");

  * n es el manejador del nodo.

 ros::NodeHandle n;

  * Mediante la función subscribe() se indica a ROS que se quieren recibir los mensajes del topic “chatter”
  * Los mensajes se pasan a una función de callback (aquí llamada chatterCallback)
  * El número 1000 indica el tamaño de la cola de mensajes recibidos.

 ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

  * ros::spin() hace que se entre en un polling buscando mensajes. Cuando llega un mensaje al topic, se llama a
la función
  * chatterCallback() indicada antes.

 ros::spin();

 return 0;
}
```

Para crear este nodo en el paquete, se debe editar el archivo “CMakeLists.txt” del directorio del paquete y añadir al final de este la siguiente línea:

rosbuild_add_executable(listener src/listener.cpp)

Esto creará un ejecutable “listener” del archivo listener.cpp que estará ubicado en la carpeta src dentro del paquete sobre el que se está trabajando.

Por último se ejecuta el comando “make” para compilar los nodos del paquete.

Para probarlos, se deben abrir tres terminales. En uno se ejecuta roscore, en otro el publicador, y en el último el suscriptor:

roscore

rosrun beginner_tutorials talker

rosrun beginner_tutorials listener

El nodo publicador empezará a mostrar la información enviada por pantalla, tal como aparece en la Fig. 4.8.

```
[INFO] [WallTime: 1314931831.774057] hello world 1314931831.77
[INFO] [WallTime: 1314931832.775497] hello world 1314931832.77
[INFO] [WallTime: 1314931833.778937] hello world 1314931833.78
[INFO] [WallTime: 1314931834.782059] hello world 1314931834.78
[INFO] [WallTime: 1314931835.784853] hello world 1314931835.78
[INFO] [WallTime: 1314931836.788106] hello world 1314931836.79
```

Figura 4.8 Datos arrojados por pantalla por el publicador.

Y el suscriptor la recibida, como puede verse en la Fig. 4.9.

```
[INFO] [WallTime: 1314931969.258941] /listener_17657_1314931968795I heard hello
world 1314931969.26
[INFO] [WallTime: 1314931970.262246] /listener_17657_1314931968795I heard hello
world 1314931970.26
[INFO] [WallTime: 1314931971.266348] /listener_17657_1314931968795I heard hello
world 1314931971.26
[INFO] [WallTime: 1314931972.270429] /listener_17657_1314931968795I heard hello
world 1314931972.27
[INFO] [WallTime: 1314931973.274382] /listener_17657_1314931968795I heard hello
world 1314931973.27
[INFO] [WallTime: 1314931974.277694] /listener_17657_1314931968795I heard hello
world 1314931974.28
[INFO] [WallTime: 1314931975.283708] /listener_17657_1314931968795I heard hello
world 1314931975.28
```

Figura 4.9 Datos arrojados por pantalla por el suscriptor.

USANDO ROSLAUNCH

El comando “roslaunch” sirve para ejecutar varios nodos al mismo tiempo. Lo primero que hay que hacer es crear un archivo de lanzamiento, por ejemplo para los tutoriales de ROS turtlemimic.launch. Su contenido será el siguiente:

```
<launch>

 <group ns="turtlesim1">
```

```
  <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
 </group>

 <group ns="turtlesim2">
  <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
 </group>

 <node pkg="turtlesim" name="mimic" type="mimic">
  <remap from="input" to="turtlesim1/turtle1"/>
  <remap from="output" to="turtlesim2/turtle1"/>
 </node>

</launch>
```

La etiqueta <launch> identifica el archivo .xml como de tipo launch. Se ve también que se inician dos grupos con espacio de nombres denominado “turtlesim1” y “turtlesim2”. En realidad se están lanzando 2 nodos turtlesim, cada uno llamado de forma distinta. Con estas definiciones se ejecutan dos nodos sin conflictos de nombres. El último bloque del archivo inicia el nodo “mimic”, que se encargará de recibir datos de un nodo turtlesim y enviar estos datos al otro nodo turtlesim para que ejecute las mismas acciones que el primero.Se le deben pasar los datos de entrada y salida, es decir los dos nodos, que son “turtlesim1” y “turtlesim2”.

Ahora, para lanzar el archivo se debe ejecutar el siguiente comando:

roslaunch [carpeta donde esta el archivo launch] [nombre del ejecutable]

Para el caso de los tutoriales de ROS será:

roslaunch beginner_tutorials turtlemimic.launch

GRABANDO Y REPRODUCIENDO DATOS

Por último, se va a ver como grabar datos para poder reproducirlos después. Esto puede ser útil para probar ciertos programas sin tener que estar conectado a un determinado sensor. Con tener una grabación de datos de este sensor, se puede trabajar sin él.

Para esta tarea se usa el comando rosbag, el cual creará archivos .bag en los que guardará los datos de uno o más topics que estén activos en el momento de la grabación. Por ejemplo, si se quiere grabar todos los topics en activo en un momento determinado, se usa el comando “rosbag record” y se añade -a para indicar que se quiere grabar la información de todos los topics:

rosbag record -a

Si solo se está interesado en determinados topics se añade -O subset y el nombre del topic o los topics que interesan. Por ejemplo, para grabar el topic en el que publica una determinada cámara, habrá que ejecutar:

rosbag record -O subset /camera/image_raw

Para reproducir los datos guardados se usa el comando "rosbag play" seguido del nombre del archivo que interesa reproducir. Para ello, habrá que posicionarse dentro de la carpeta en la que esta guardado el archivo .bag. Si se añade -r [número], se puede cambiar la velocidad de reproducción en un factor igual al número que elijamos, tanto aumentarla como disminuirla, y si se añade –l, se puede reproducir el archivo en bucle infinito. Por ejemplo, para reproducir una grabación de nombre camaraestereo de manera normal, la misma 10 veces más lenta, y la grabación en bucle infinito se ejecutarán respectivamente los comandos:

rosbag play camaraestereo.bag

rosbag play -r 0.1 camaraestereo.bag

rosbag play -l camaraestereo.bag

4.2 ROS en el proyecto

Una vez que se conoce el funcionamiento básico de ROS, se pasa a analizar los programas usados en el proyecto:

4.2.1 Sensor inercial

Se trata de un sensor inercial modelo IMU microstrain 3dmgx2, que tiene su propio paquete en los repositorios de ROS para facilitar su uso [30]. Para poder usar este paquete hay que tener instalada la versión hydro de ROS y seguir los siguientes pasos para la instalación del paquete IMU.

Hay que empezar descargando el paquete y compilarlo con los siguientes comandos:

rosdep install microstrain_3dmgx2_imu

rosmake microstrain_3dmgx2_imu

Ahora se debe conectar el sensor inercial por USB y lanzar roscore en un terminal.

A continuación, en un nuevo terminal se ejecuta el nodo del sensor mediante el comando:

rosrun microstrain_3dmgx2_imu imu_node

Si esto provoca un error, puede ser debido a que el sensor no esta conectado al USB principal "/dev/ttyUSB0". Si esto sucede, lo que hay que hacer es lanzar el siguiente comando con el número de puerto correcto, por ejemplo, si tenemos el sensor conectado al puerto USB 1:

rosrun microstrain_3dmgx2_imu imu_node _port:=/dev/ttyUSB1

Ahora, para ver los datos publicados por el sensor inercial, en un nuevo terminal se ejecuta:

rostopic echo imu/data

Introduciendo este comando, el terminal empieza a mostrar los datos que llegan desde el sensor, los cuales se pueden ver en la Fig. 4.10.

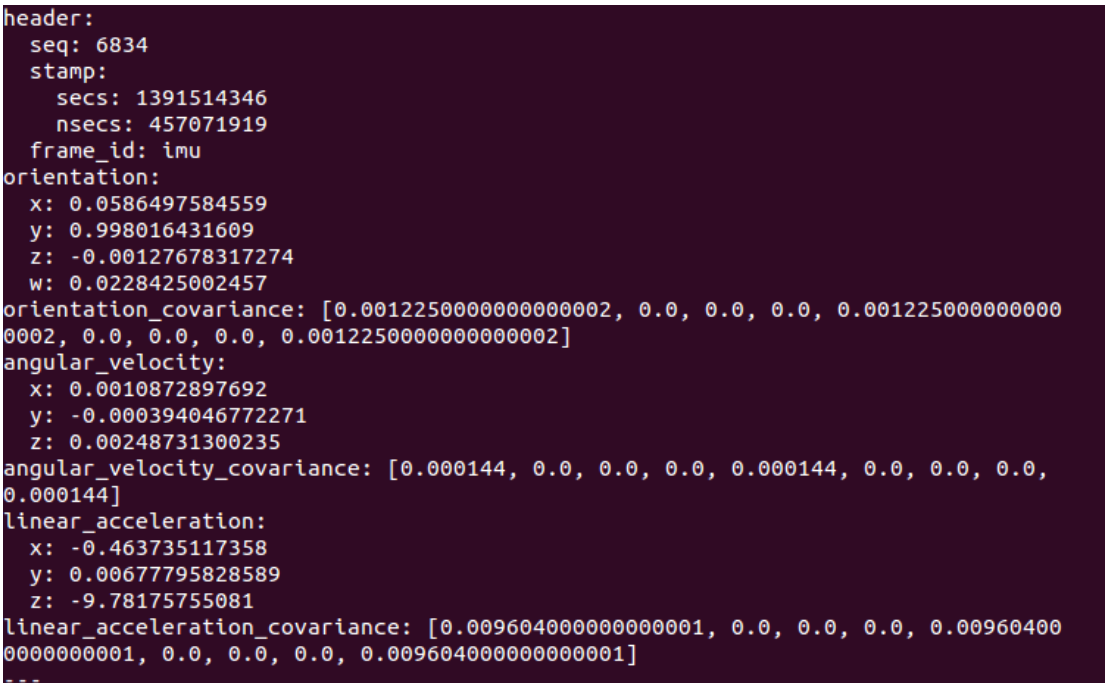
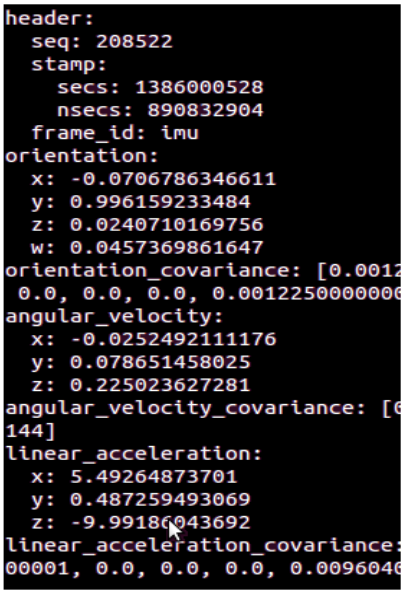


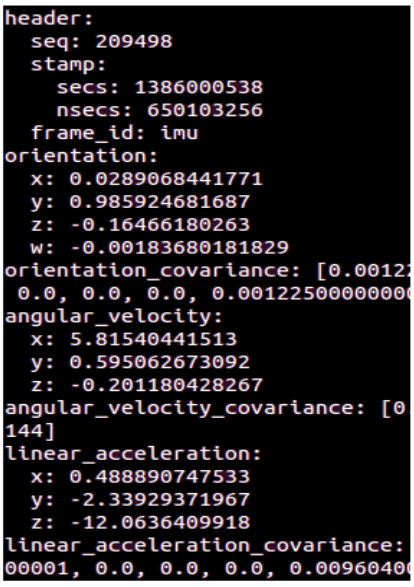
Figura 4.10 Datos arrojados en el terminal usando imu/data.

Estos datos son la orientación, la velocidad angular, la aceleración lineal y datos relacionados con la conexión como el puerto de conexión. La orientación viene dada en radianes(rad), la velocidad angular en radianes por segundo(rad/s), y la aceleración lineal en metros por segundo cuadrado(m/s^2). En la aceleración del eje z se tiene un valor de aproximadamente -9,8 m/s^2 y que corresponde a la aceleración de la gravedad. Para la orientación se tiene un cuaternio de valores compuesto por un vector de tres valores x, y, z y por un escalar w.

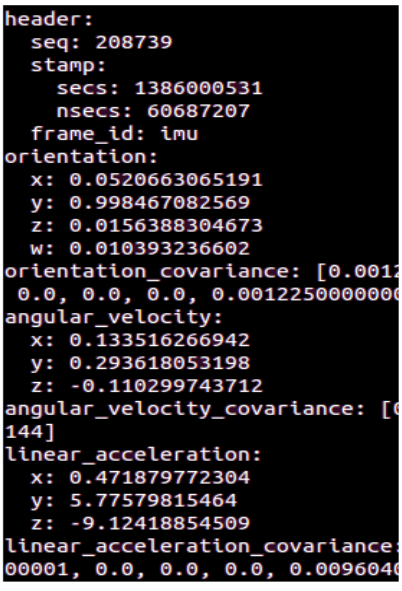
Las siguientes imágenes muestran varios ejemplos de los resultados obtenidos con el programa al aplicar distintos movimientos sobre el sensor inercial. En las figuras Fig. 4.11 (a), (c), y (e), se aplica un desplazamiento lineal sobre el eje X, Y y Z respectivamente. En las figuras Fig.4.11 (b), (d), y (f), se aplica una rotación sobre el eje X, Y y Z respectivamente. Se puede ver como la aceleración lineal en el eje Z tiene en cuenta la aceleración de la gravedad.



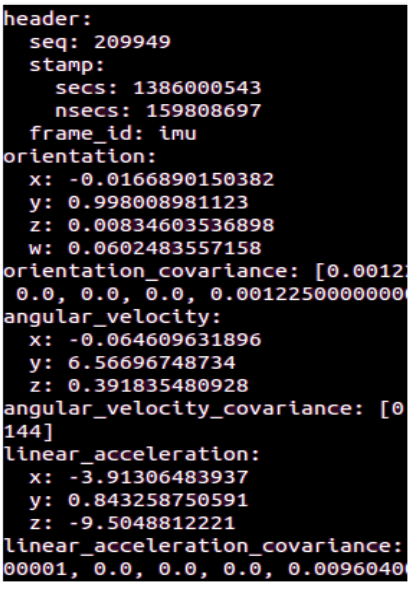
(a)



(b)



(c)



(d)

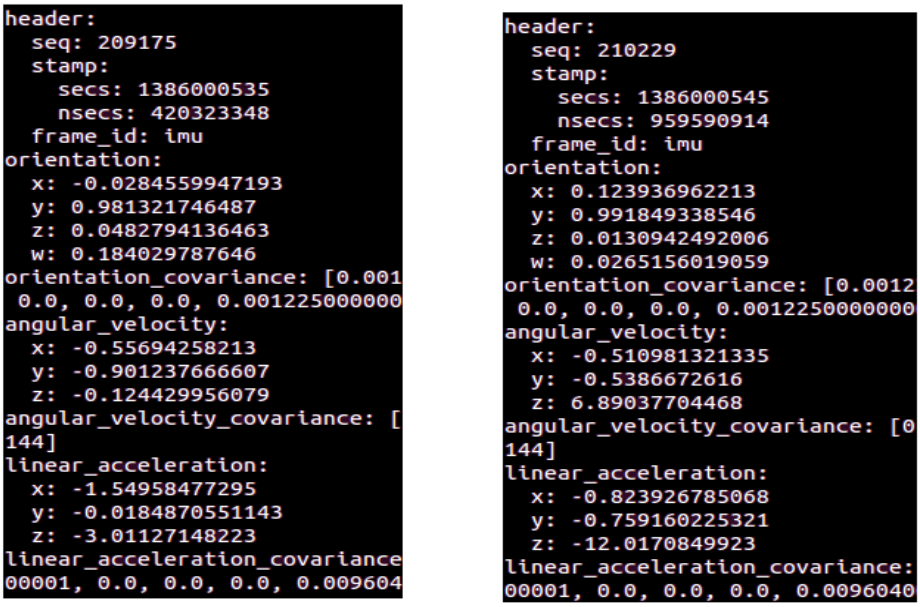


Figura 4.11 Datos del sensor inercial para los distintos movimientos:(a), (c) y (e) lineal en x, y, z; (b), (d) y (f) rotación sobre x, y, z.

Además de este tópico que muestra los valores tomados por el sensor inercial, se tienen otros dos tópicos, que son diagnostics, el cual envía información sobre el estado del sensor, e imu/is_calibrated, que indica si el sensor ha sido calibrado.

Aparte de los tópicos, tenemos otros servicios disponibles dentro del paquete. Estos son self_test, que sirve para iniciar un test de prueba del sensor, e imu/calibrate, que sirve para recalibrar los ejes del sensor.

4.2.2 Cámaras estéreo y mono

Para este proyecto se ha usado una camara estéreo modelo Bumblebee2 de Pointgrey. Se ha utilizado una biblioteca propia que toma las imágenes de las dos cámaras y después de aplicar las operaciones necesarias sobre estas, publica un tópico para la imagen calibrada y rectificada y otros dos con las imágenes de las cámaras derecha e izquierda por separado. Esto se hace con el programa “bumblebee2.cpp” incluido en el apéndice. En la Fig. 4.12 se pueden ver los tópicos arrojados por este programa.

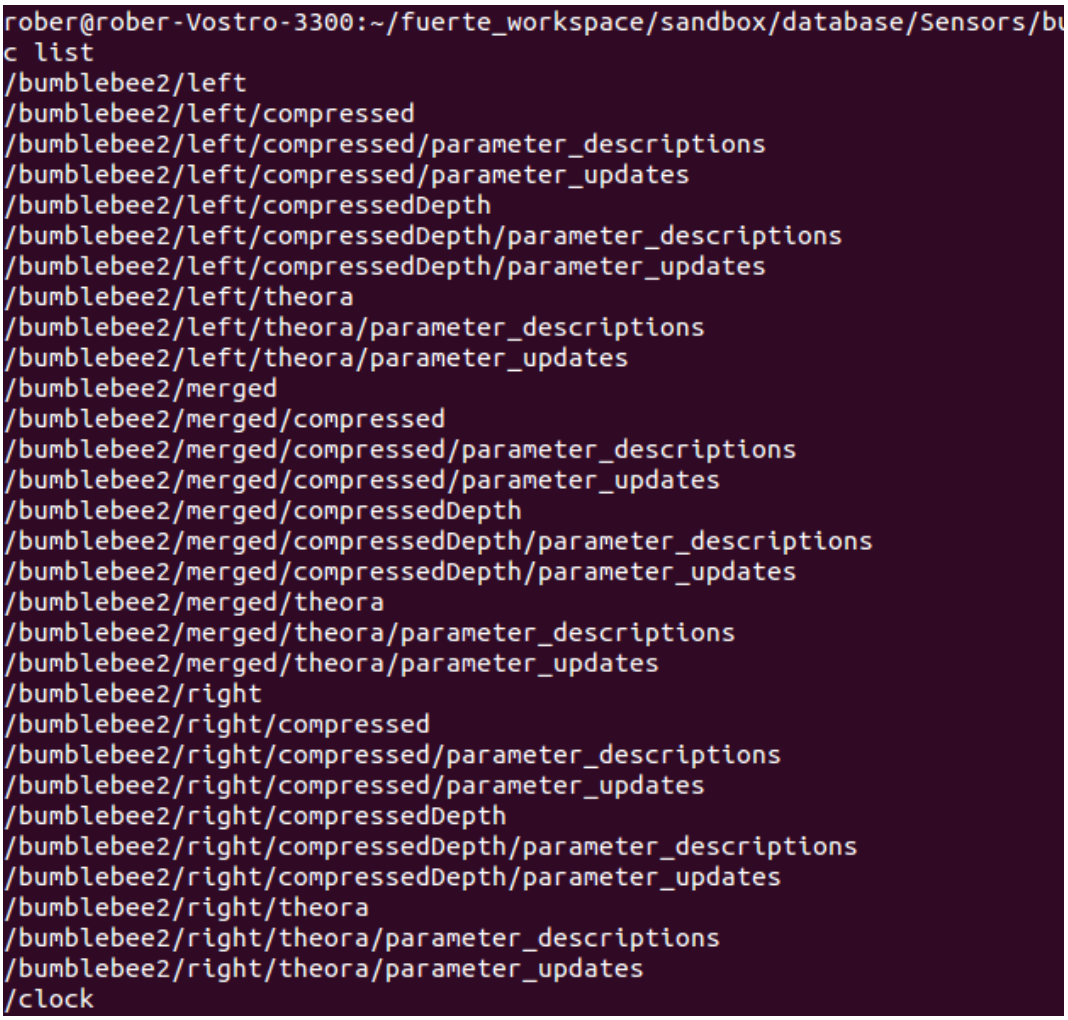


Figura 4.12 Tópicos arrojados al usar bumblebee2.

A continuación, con otro programa, “camestereo.cpp”, tambien incluido en el apéndice, se toma una de las imágenes publicadas y tratadas por el programa anterior, en este caso la del lado derecho, “/bumblebee2/right”, y aplicando una serie de operaciones y mediante el método de histograma de gradientes orientados, se publican dos tópicos, uno con una imagen en la que se recuadran los peatones detectados en la imagen (Fig. 4.13), “/estereo/image”, y otro con las coordenadas de las posiciones de los peatones en la imagen (Fig. 4.14), “datoscamestereo”, que muestra las coordenadas x e y de la esquina superior izquierda de la posición del peatón, el ancho en píxeles del peatón detectado, y el alto también en píxeles del peatón detectado, es decir, la posición del peatón en la imagen.

La parte más complicada de este programa consiste en transformar los datos para que puedan ser procesados utilizando comandos de las bibliotecas Open CV y la operación inversa, para poder enviar estos datos procesados a través de un topic.

Esto se consigue utilizando el objeto CvBridge incluido dentro de ROS, el cual permite, mediante su método imgMsgToCv tomar una imagen que está siendo procesada por ROS y transformarla en un objeto imagen del tipo Iplimage, que es el usado en Open Cv. Una vez se tiene la imagen en un formato procesable mediante Open Cv, se aplica el método de HOG para la detección de peatones. Una vez que la imagen ha sido procesada, se puede enviar directamente el resultado al topic, ya que no se tendrá que modificar utilizando métodos propios de ROS.

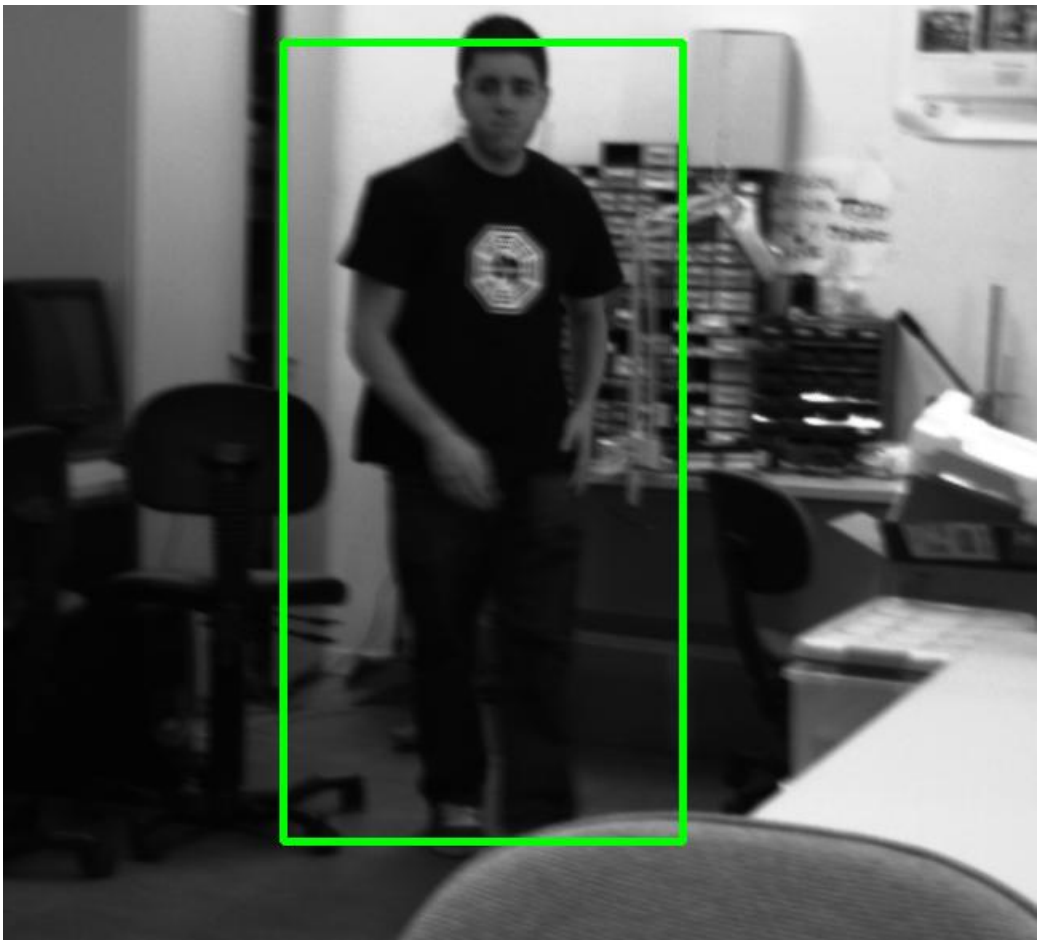


Figura 4.13 Ejemplo de imagen enviada al tópico estereo/Image.

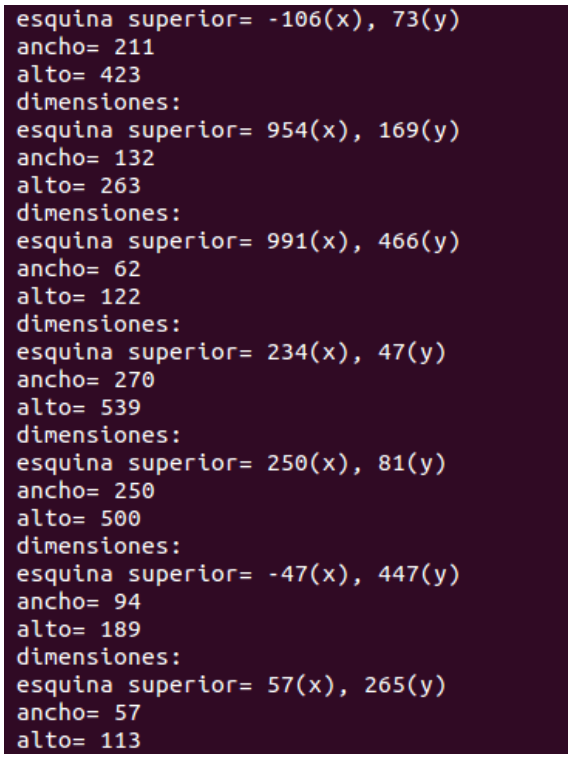


Figura 4.14 Ejemplo de datos enviados al tópico datoscamestereo.

La cámara mono usada, ha sido una cámara 1394 estándar. Se han utilizado las bibliotecas disponibles en los repositorios de ROS para este tipo de cámaras y un programa muy parecido al camestereo.cpp, que en este caso se llama "cammono.cpp". El primer paso es instalar el paquete "camera 1394" y compilarlo [31] mediante los comandos:

rosdep install camera1394

rosmake camera1394

A continuación se conecta la cámara. Para probar que la cámara funciona, se puede instalar "coriander", que es un programa que permite ver la imagen captada por una cámara si esta esta conectada al ordenador, y ver si se recibe la imagen. Esto se hace mediante el comando:

sudo apt-get install coriander

Para iniciar la toma de datos de la cámara se usa el siguiente comando:

rosrun camera1394 camera1394_node

Los tópicos publicados que interesan son:

/camera/camera_info

/camera/image_raw

/camera_info publica datos sobre la imagen de la cámara, como los parámetros intrínsecos. /image_raw publica la imagen.

Para ver la imagen, primero se compila image_view:

rosmake image_view

Y después, desde este programa, hay que suscribirse al tópico publicado:

rosrun image_view image_view image:=camera/image_raw

La cámara permite muy diversas configuraciones de la imagen, que pueden ser configuradas antes de iniciar el programa de captura o de manera dinámica usando el comando:

rosrun rqt_reconfigure rqt_reconfigure

Para obtener más información sobre la configuración de las cámaras 1394, se puede consultar el tutorial de **http://wiki.ros.org/camera1394**.

Después de iniciar la toma de imágenes y los tópicos necesarios de la cámara con este programa, se inicia el programa cammono.cpp. Este se suscribe a la imagen publicada por el anterior programa, /image/image_raw, y aplica el mismo procedimiento que el usado para la cámara estéreo, solo que en este caso los tópicos publicados son, /mono/Image, que muestra una imagen con los peatones detectados recuadrados, y datoscammono que envía un mensaje con las coordenadas de la posición de los peatones detectados.

4.2.3 GPS

Para el proyecto se ha usado un modelo de GPS que puede ser configurado para trabajar de muy diversas formas. En este caso, esta configurado para que trabaje tomando datos con una frecuencia de 1 Hz. y con un servicio de posicionamiento estándar. Este modo de funcionamiento se encuentra dentro de los definidos por el protocolo NMEA (National Marine Electronics Association). Dentro de este protocolo se tienen distintos modos de posicionamiento, entre los que varía la precisión de la medida. Estos modos son [32 y 33]:

SPS(Standard Positioning Service):

Se trata de un servicio de posicionamiento estándar. El funcionamiento del GPS con este método es muy sencillo. En cualquier lugar del mundo un usuario puede determinar su posición geográfica en tres dimensiones con gran exactitud durante las 24 horas del día, independientemente de las condiciones meteorológicas, mediante el cálculo de las distancias entre la antena del receptor GPS y los satélites

que tenga a la vista. En principio, con cuatro satélites es suficiente para obtener las coordenadas (x, y, z, t), ya que, aunque haya dos posibles soluciones para una determinada posición, de los dos puntos teóricamente posibles, uno es absurdo, es decir, no es físicamente posible. Sin embargo, como dicho cálculo se basa en la medición del tiempo que tarda en llegar al receptor la señal de cada satélite, las distancias determinadas de esta manera se ven afectadas por el error de sincronización entre el transmisor y el receptor. La sincronización de los satélites entre sí se resuelve acoplando en ellos relojes atómicos de gran precisión, pero los relojes de los receptores no son de tanta calidad, y esto sigue dando lugar a un error en la medida.

Los principales errores que se producen y que influyen en la exactitud final con que se puede determinar la posición a partir de la señal GPS son los derivados de:

- los relojes de los satélites y la desviación de la órbita, que han de solucionarse desde el control de tierra.

- la transmisión de las señales a través de la ionosfera, que se resuelven casi por completo teniendo en cuenta que la pérdida de velocidad de las ondas de radio a través de esta es inversamente proporcional al cuadrado de la frecuencia empleada.

- la transmisión de las señales a través de la troposfera, casi imposibles de corregir.

- los ruidos de los receptores, que dependen de la calidad de éstos.

- el efecto multitrayectoria, ocasionado por las reflexiones sobre determinados obstáculos de las señales de los satélites antes de llegar al receptor, efecto que debe evitarse mediante los estudios apropiados.

- la incertidumbre geométrica, debida a la posición relativa de los satélites empleados para la determinación de la posición, que se minimiza mediante la selección de los satélites más adecuados en cada ocasión.

DGPS(Diferencial GPS):

El DGPS es un sistema mejorado que intenta paliar los inconvenientes derivados de los errores enumerados en el sistema GPS y conseguir una exactitud e integridad mejoradas.

Basado en las señales del GPS y con estaciones de referencia en tierra cuya posición es conocida, calcula y transmite las correcciones que los usuarios han de aplicar a los datos GPS para obtener una posición más exacta dentro de la zona cubierta por las emisoras.

El fundamento de este sistema consiste en que la estación de referencia determina su posición a partir de las señales GPS y, comparándola con su posición conocida, calcula las diferencias o correcciones que deben aplicarse a los resultados obtenidos a partir de los satélites para que ambas posiciones coincidan. Estas correcciones son

las que se transmiten a los usuarios del sistema, cuyos equipos DGPS las introducen en sus cálculos para determinar la posición.

Esta forma de operar es válida si la estación de referencia y el usuario están a unos pocos centenares de kilómetros, ya que de esta manera las señales GPS que lleguen a ambos, habrán atravesado zonas del espacio con unas características prácticamente idénticas y las correcciones a aplicar para obtener una medida más precisa serán casi las mismas.

Esta limitación en el alcance hace que a este tipo de sistema de posicionamiento se le llame de cobertura local.

PPS(Precise Positioning Service):

Este modo es más preciso que el SPS y en un principio era solo de uso militar, pero ahora se puede usar si se dispone de un receptor compatible con este modo de funcionamiento. Utiliza bandas de frecuencia que producen menos error.

RTK(Real Time Kinematic):

El modo RTK(navegación cinética satelital en tiempo real), se basa en el uso de medidas de fase de navegadores con señales GPS, donde una sola estación de referencia proporciona correcciones en tiempo real, obteniendo una exactitud submétrica. Los receptores comunes basados en la navegación por satélite, comparan una señal pseudoaleatoria que es enviada desde el satélite con una copia interna generada por la misma señal. Puesto que la señal del satélite tarda un tiempo en alcanzar al receptor, las dos señales no se sincronizan correctamente; se empieza a retrasar la copia local en referencia a la copia del satélite. Al retrasar progresivamente la copia local, las dos señales se alinearán correctamente en algún momento. Este retraso es el tiempo necesario para que la señal alcance al receptor, y del resultado de esto puede ser calculada la distancia al satélite. La precisión de la medición resultante es generalmente una función de la capacidad electrónica del receptor para comparar las dos señales.

Float RTK:

En este caso, la solución arrojada por el GPS puede ser un número decimal o un número en coma flotante. El receptor puede trabajar sin que haya un número suficiente de satélites en común con la estación base para ajustar la medida [34].

En la Fig. 4.15. se puede ver la diferencia de errores producidos, entre los distintos modos de funcionamiento.

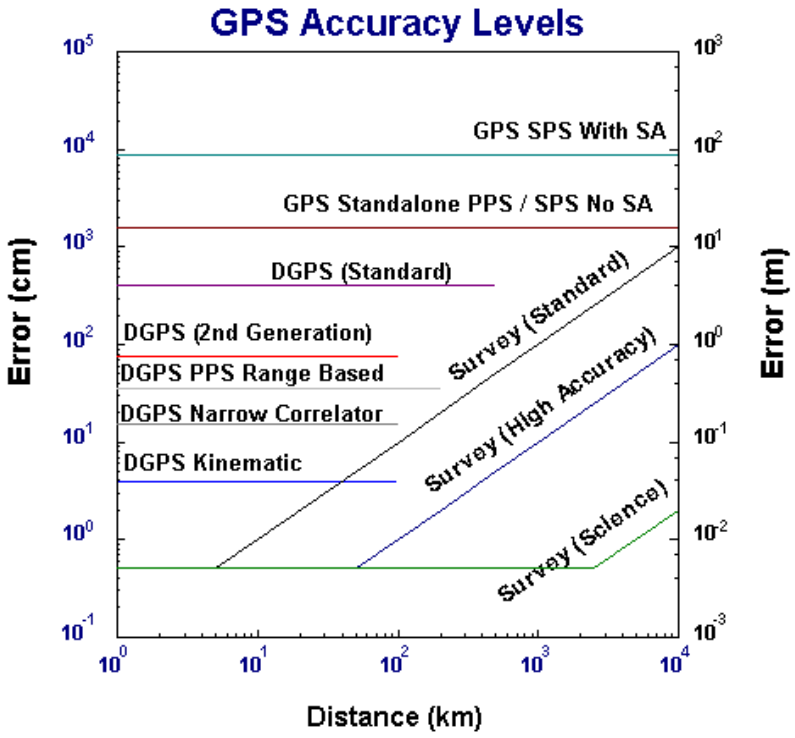


Figura 4. 15 Precisión según el tipo de GPS.

Para explicar el significado de cada dato en una trama del protocolo NMEA que hemos usado, vamos a coger la siguiente trama, que es una trama enviada por el GPS, y dividirla por partes:

$ GPGGA, 133505.10, 4020.0134, N, 00345.9988, W, 1, 04, 5.2, 657.98, M, 52.10, M, , 7C

Todas las tramas empiezan con **$ GP** para identificar al código como una trama enviada por el GPS.

GGA -> Indica que se utiliza un sistema de posicionamiento global.

133505.10 -> Muestra la hora, minuto, segundo y centésima de segundo en los que se han tomado los datos de la trama.

4020.0134,N -> Indica la latitud en la que se encuentra el GPS. En este caso, latitud 40, 20,0134 grados norte.

00345.9988,W -> Indica la longitud en la que esta situado el GPS. En este caso, longitud 3, 45.9988 grados oeste.

1 -> Indica la calidad de la medida. Esta puede ser de diferentes tipos:

-0 inválido.

-1 SPS (Standard Positioning Service).

-2 DGPS (diferencial).

-3 PPS (Precise Positioning Service).

-4 RTK (Real Time Kinematic, navegación cinemática satelital).

-5 Float RTK (RTK flotante).

-6 estimado (dead reckoning).

-7 modo de introducción manual.

-8 modo de simulación.

En el caso de ejemplo, se trabaja en un modo de calidad estándar.

04 -> Número de satélites monitorizados. En este caso 4.

5.2 -> Dilución de la Precisión (DOP). Es una medida de la fortaleza de la geometría de los satélites y está relacionada con la distancia entre los satélites y su posición en el cielo. El DOP puede incrementar el efecto del error en la medición de distancia a los satélites. En este caso es 5,2.

657.98,M -> Altitud en metros sobre el nivel medio del mar, que en este caso es 657,98.

52.10,M -> Altura del geoide sobre el elipsoide WGS84.

Si el GPS trabajara en modo diferencial, se tendría otro valor de 4 cifras, el cual indicaría el número de identificación de la estación con la cual se corrige la posición.

El valor **7C** es un valor de comprobación, que depende de los valores anteriores. Así, se puede comprobar que los datos han llegado bien para empezar a procesarlos.

Para empezar a trabajar con el GPS, primero hay que instalar el paquete cereal_port [35], el cual permite gestionar la comunicación en serie entre el ordenador y los periféricos de una manera más sencilla. Para ello hay que descargar el paquete cereal_port disponible en la Wiki de ROS. Una vez descargado, lo que hay que hacer es crear un programa que permita leer los datos recibidos por el puerto USB en el que se conecta el GPS. El código de este programa se muestra en la parte de apéndice de este manual. El programa tambien tiene que tomar los datos recibidos desde el GPS y lanzar un tópico con la trama recibida para que otro programa pueda procesar estos datos si es necesario.

Para probar el correcto funcionamiento del programa, se conecta el GPS por USB al PC y se configura el nombre del puerto de entrada dentro del programa que se ha creado. Para conocer este nombre de una manera rápida y fácil, se puede instalar el programa “serial port terminal”, el cual permite ver los periféricos conectados en serie al ordenador y el nombre de puerto al que esta conectado cada uno. Además,

como permite ver los datos enviados y recibidos entre PC y periférico, se puede comprobar que el programa toma y reenvía los datos de una manera correcta.

Teniendo roscore en funcionamiento, y teniendo en cuenta que el nombre del programa es gps y que esta contenido en la carpeta bumblebee2, hay que ejecutar el siguiente comando para poner en marcha el programa:

rosrun bumblebee2 gps

Un ejemplo de lectura continua con el programa, haciendo una pequeña modificación en este para que se muestren los datos leidos por pantalla es el de la Fig. 4.16, en el que se muestra los datos que son enviados mediante el tópico datosgps.

```
[ INFO] [1401552997.885588201]: $GPGGA,133505.10,4020.0134,N,00345.9988,W,1,04,5
.2,657.98,M,52.10,M,,*7C
[ INFO] [1401552998.885605148]: $GPGGA,133505.20,4020.0132,N,00345.9991,W,1,04,5
.2,657.69,M,52.10,M,,*7F
[ INFO] [1401552999.885578993]: $GPGGA,133505.30,4020.0130,N,00345.9994,W,1,04,5
.2,657.76,M,52.10,M,,*77
[ INFO] [1401553000.885595143]: $GPGGA,133505.40,4020.0127,N,00345.9997,W,1,04,5
.2,657.51,M,52.10,M,,*70
```

Figura 4.16 Datos arrojados por el GPS.

4.2.4 Funcionamiento conjunto de todos los sensores

Una vez que se ha comprobado que todos los sensores funcionan correctamente, el siguiente paso es ponerlos a funcionar a la vez. La complicación de hacerlo, está en que la cámara y el GPS funcionan con la distribución fuerte de ROS, y el sensor inercial funciona con la distribución hydro de ROS. Para conseguir que funcionen a la vez se deberá configurar la fuente de la que se toman los datos (carpeta de una distribución o de la otra), usando el comando adecuado. Para ello se tienen los siguientes comandos:

source ~/fuerte_workspace/setup.bash

Este comando hace que se pueda trabajar con los programas que se tienen guardados en una carpeta específica, en este caso"fuete_workspace", que funciona con la distribución fuerte de ROS.

source /opt/ros/hydro/setup.bash

Con este otro se pasa a trabajar en el directorio principal de la distribución hydro de ROS.

Para cada programa se abrirá un terminal. En el caso de la cámara y el GPS se utilizará el primer comando y en el caso del inercial se utilizará el segundo.

Una vez ejecutados los programas, se empiezan a recibir los datos de todos los sensores y se envian para poder procesarlos.

Todos estos pasos se podrían ejecutar en un archivo .launch para evitar toda la secuencia de comandos realizada y poder poner todos los sensores en ejecución con un solo comando.

En el próximo capítulo se verán el cálculo estimado de los costes del proyecto y las conclusiones del mismo.

5. Costes del proyecto y conclusiones

5.1 Costes

El objetivo de este capítulo es presentar una relación estimada de los costes del proyecto. Teniendo en cuenta unos tiempos aproximados para cada fase del proyecto de:

FASE	TIEMPO
Estudio y comprensión del problema a resolver	12 horas
Documentación	25 horas
Implementación del código	80 horas
Pruebas y correciones del código	25 horas
Redacción de la memoria	60 horas
TOTAL	**202 horas**

Suponiendo que el coste aproximado de un programador junior es de de 30 €/hora (impuestos incluidos, el precio de la mano de obra sería:

$$202\ horas * 30\ €/\text{hora} = 6060\ €$$

Además, debemos tener en cuenta el coste por materiales, que podemos dividir en:

CONCEPTO	COSTE
PC estándar	1000 €
Cámara estéreo (Bumblebee 2 de Pointgrey)	3000 €
Sensor inercial Microstrain 3dm-gx2	1200 €
GPS	6000 €
Otros materiales (soprtes, cables…)	100 €
TOTAL	**11300** €

Si suponemos que se tardan 5 horas en realizar el montaje sobre el automóvil, y que el operarioque se encarga de ello cobra 30 €/hora, la instalación cuesta:

$$5\ horas * 30\ €/\text{hora} = 150\ €$$

El coste total de proyecto queda como:

$$6060\,€ + 11300\,€ + 150\,€ = \mathbf{17510\,€}$$

5.2 Conclusiones

El objetivo principal de este proyecto era la implementación de distintos sensores en un coche y la detección de peatones mediante el sistema ROS. Esto se ha conseguido mediante la creación de diversos programas y la utilización de código ya existente en los repositorios ofrecidos en la página oficial de ROS. Además, se han creado los tópicos correspondientes para cada sensor, de manera que los datos recogidos por estos puedan ser utilizados por otros programas o proyectos futuros.

5.3 Futuros trabajos

Para seguir avanzando en este proyecto, se podría implementar un sistema que calibrara automáticamente la cámara mono y la estéreo, de manera que se pudieran utilizar los datos de la relación entre ambas para conseguir información adicional como la distancia a la que se encuentra un peatón determinado.

Por otro lado se podría mejorar la eficiencia del detector de peatones, ya que el objetivo de este proyecto era conseguir la detección y la implementación de todos los sensores en conjunto, y no importaba la eficiencia del proceso.

Apéndice

A. Código del programa de referencia de OpenCV para la detección de peatones

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdio.h>
#include <string.h>
#include <ctype.h>

using namespace cv;
using namespace std;

int main(int argc, char** argv)
{
Mat img;
FILE* f = 0;
char _filename[1024];
if( argc == 1 )
{
printf("Usage: peopledetect (<image_filename> | <image_list>.txt)\n");
return 0;
}
img = imread(argv[1]);
if( img.data )
{
strcpy(_filename, argv[1]);
}
else
{
```

```
f = fopen(argv[1], "rt");
if(!f)
{
fprintf( stderr, "ERROR: the specified file could not be loaded\n");
return -1;
}
}
HOGDescriptor hog;
hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());
namedWindow("people detector", 1);
for(;;)
{
char* filename = _filename;
if(f)
{
if(!fgets(filename, (int)sizeof(_filename)-2, f))
break;
if(filename[0] == '#')
continue;
int l = (int)strlen(filename);
while(l > 0 && isspace(filename[l-1]))
--l;
filename[l] = '\0';
img = imread(filename);
}
printf("%s:\n", filename);
if(!img.data)
continue;
fflush(stdout);
vector<Rect> found, found_filtered;
double t = (double)getTickCount();
// run the detector with default parameters. to get a higher hit-rate
// (and more false alarms, respectively), decrease the hitThreshold and
// groupThreshold (set groupThreshold to 0 to turn off the grouping completely).
```

```
hog.detectMultiScale(img, found, 0, Size(8,8), Size(32,32), 1.05, 2);
t = (double)getTickCount() - t;
printf("tdetection time = %gms\n", t*1000./cv::getTickFrequency());
size_t i, j;
for( i = 0; i < found.size(); i++ )
{
Rect r = found[i];
for( j = 0; j < found.size(); j++ )
if( j != i && (r & found[j]) == r)
break;
if( j == found.size() )
found_filtered.push_back(r);
}
for( i = 0; i < found_filtered.size(); i++ )
{
Rect r = found_filtered[i];
// the HOG detector returns slightly larger rectangles than the real objects.
// so we slightly shrink the rectangles to get a nicer output.
r.x += cvRound(r.width*0.1);
r.width = cvRound(r.width*0.8);
r.y += cvRound(r.height*0.07);
r.height = cvRound(r.height*0.8);
rectangle(img, r.tl(), r.br(), cv::Scalar(0,255,0), 3);
}
imshow("people detector", img);
int c = waitKey(0) & 255;
if( c == 'q' || c == 'Q' || !f)
break;
}
if(f)
fclose(f);
return 0;
}
```

B. Código de los programas usados en ROS

B.1 Código para el GPS

```
#include <ros/ros.h>
#include <stdio.h>
#include <iostream>
#include "std_msgs/String.h"
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <cv_bridge/CvBridge.h>
#include <cvaux.h>
#include <math.h>
#include <cxcore.h>
#include <highgui.h>
#include <fstream>
#include <string.h>
#include <cereal_port/CerealPort.h>
//definimos todos los paquetes necesarios
#define REPLY_SIZE 100 //definimos unas constantes que nos seran utiles
#define TIMEOUT 100
int main(int argc, char** argv)
{
ros::init(argc, argv, "gps"); //inicializamos ROS y le damos nombre al programa
ros::NodeHandle n;
ros::Publisher chatter_pub = n.advertise<std_msgs::String>("datosgps", 1000);
//definimos el topico en que publicaremos
cereal::CerealPort gps; //creamos un tipo de dato que sirve para tomar datos por el
puerto serie
```

```
char captura[REPLY_SIZE]; //creamos un caracter que nos servira para la publicacion de datos y le damos valor inicial
captura[0]='\0';
try{ gps.open("/dev/ttyACM0", 9600); } //definimos el puerto en el que esta conectado el GPS y el baud rate al que se produce la comunicacion
catch(cereal::Exception& e) //abrimos el puerto serie
{
ROS_FATAL("Error al abrir el puerto serie!!!"); //si no se abre correctamente, enviamos un aviso
ROS_BREAK();
}
ROS_INFO("El puerto serie esta abierto."); //si no, enviamos el mensaje de confirmacion
ros::Rate r(1);// intervalo de envio //definimos la frecuencia de envio a 1 hz
while(ros::ok())
{
try{ gps.read(captura,TIMEOUT); } //tomamos los datos recibidos por el puerto serie
catch(cereal::TimeoutException& e)
{
ROS_ERROR("Timeout!");
}
captura[73]='\0'; //tomamos solo los datos que nos interesan segun el protocolo usado para envio de datos del GPS
std_msgs::String msg;
std::stringstream ss; //publicamos los datos recogidos en el topico abierto
ss << captura;
msg.data = ss.str();
ROS_INFO("%s", msg.data.c_str());
chatter_pub.publish(msg);
ros::spinOnce();
r.sleep();
}
}
```

B.2 Código para la detección de peatones en la cámara mono

```
#include <ros/ros.h>
#include <stdio.h>
#include <iostream>
#include "std_msgs/String.h"
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <cv_bridge/CvBridge.h>
#include <cvaux.h>
#include <math.h>
#include <cxcore.h>
#include <highgui.h>
#include <fstream>
#include <string.h>
//incluimos todos los paquetes necesarios para el funcionamiento del programa
using namespace std;
using namespace cv;
namespace enc = sensor_msgs::image_encodings;
static const char WINDOW[] = "Image window"; //habilitamos una ventana para mostrar la imagen
class detect
{
ros::NodeHandle nh_; //definimos los datos para poder comunicarnos con otro programa
ros::NodeHandle n;
ros::Publisher pub;
image_transport::ImageTransport it_;
```

```
image_transport::Subscriber image_sub_; //nombre del tema al que nos suscribimos
image_transport::Publisher image_pub_; //nombre del tema en el que se publica
std_msgs::String msg;
public:
detect()
: it_(nh_)
{
pub= n.advertise<std_msgs::String>("datoscammono", 1000) ; //le damos nombre al topico en el que escribimos
image_sub_ = it_.subscribe("/camera/image_raw", 1, &detect::imageCb, this); //nos suscribimos al topico de la camara estereo
image_pub_= it_.advertise("/mono/Image",1); //publicamos en el topico con el nombre seleccionado
}
~detect()
{
cv::destroyWindow(WINDOW); //al acabar, cerramos la ventana habilitada
}
void imageCb(const sensor_msgs::ImageConstPtr& msg) //llamamos al programa principal
{
char xchar[5]; //definimos algunas variables que nos seran necesarias
char ychar[5];
char anchochar[5];
char altochar[5];
sensor_msgs::CvBridge bridge; //definimos la variable para la conversion de la imagen desde ROS a OpenCV
IplImage* img = bridge.imgMsgToCv(msg,"bgr8"); //convertimos la imagen desde ROS a OpenCV usando cvbridge
Mat img2(img); //copiamos la imagen para poder trabajar con el duplicado
HOGDescriptor hog; //con estas dos sentencias declaramos el uso del histograma de gradientes Orientados
hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());
vector<Rect> found, found_filtered; //aplicamos el histograma de gradientes orientados
```

```
double t = (double)getTickCount();
int can = img2.channels();
hog.detectMultiScale(img2, found, 0, Size(8,8), Size(32,32), 1.05, 2);
t = (double)getTickCount() - t;
//printf("tdetection time = %gms\n", t*1000./cv::getTickFrequency());
size_t i, j;
for (i=0; i<found.size(); i++) //aplicamos una correccion a la solucion obtenida
{
Rect r = found[i];
for (j=0; j<found.size(); j++)
if (j!=i && (r & found[j])==r)
break;
if (j==found.size())
found_filtered.push_back(r);
}
for (i=0; i<found_filtered.size(); i++)
{
Rect r = found_filtered[i]; //definimos las dimensiones del peaton
r.x += cvRound(r.width*0.1);
r.width = cvRound(r.width*0.8);
r.y += cvRound(r.height*0.07);
r.height = cvRound(r.height*0.8);
rectangle(img2, r.tl(), r.br(), cv::Scalar(0,255,0), 3); //dibujamos el rectangulo
alrededor del peaton detectado
printf("dimensiones: \n esquina superior= %i(x), %i(y) \n ancho= %i \n alto= %i \n ",
r.x,
r.y, r.width, r.height);
sprintf(xchar, "%d", r.x); //transformamos los valores de posicion del rectangulo que
contiene al peaton
sprintf(ychar, "%d", r.y); //de entero a caracter para poder publicarlo
sprintf(anchochar, "%d", r.width);
sprintf(altochar, "%d", r.height);
std_msgs::String msg; //publicamos la informacion que nos interesa
std::stringstream ss;
```

```
ss << "\ndimensiones: esquina superior izquierda= " << xchar <<"(x), " << ychar <<"(y)
ancho=" << anchochar << " alto=" << altochar;
msg.data = ss.str();
pub.publish(msg);
}
imshow("Detector", img2); //mostramos la imagen con la deteccion ya realizada y en la posicion de la pantalla que nos interesa
cvMoveWindow("Detector",200,50);
cvWaitKey(2);
}
};
int main(int argc, char** argv)
{
ros::init(argc, argv, "peopledetect"); //definimos todo lo necesario para el correcto funcionamiento de ROS
detect ic;
ros::spin();
return 0;
}
```

B.3 Códigos para la obtención de la imagen de la cámara estéreo

```
#include <ros/ros.h>
#include "ros/package.h"
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
//definimos los paquetes necesarios
cv::Mat left(768, 1024, CV_8UC1); //definimos variables que nos seran utiles y los topicos en los que publicaremos
cv::Mat right(768, 1024, CV_8UC1);
cv::Mat merged(2*768, 1024, CV_8UC1);
cv::Mat imtmp(768, 1024, CV_8UC3);
cv::vector<cv::Mat> ch;
cv::Mat remaplx, remaply, remaprx, remapry;
image_transport::Publisher pub_merged, pub_right, pub_left;
void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
cv_bridge::CvImagePtr cv_ptr; //definimos una variable para el paso de datos desde ROS a OpenCV
cv_ptr = cv_bridge::toCvCopy(msg, "bayer_grbg16"); //y transformamos la imagen que nos interesa
for(int k = 0; k<1024*768; k++)
{
left.data[k] = cv_ptr->image.data[k*2]; //separamos toda la imagen en dos partes, la del lado izquierdo y derecho
right.data[k] = cv_ptr->image.data[k*2+1];
}
cv::cvtColor(right, imtmp,CV_BayerGB2RGB); //aplicamos un filtro
cvtColor(imtmp, right, CV_RGB2GRAY);
cv::cvtColor(left, imtmp,CV_BayerGB2RGB);
cvtColor(imtmp, left, CV_RGB2GRAY);
```

```
cv::remap(right, right, remapry, remaprx, CV_INTER_LINEAR, cv::BORDER_CONSTANT, cv::Scalar()); //rectificamos las imagenes
cv::remap(left, left, remaply, remaplx, CV_INTER_LINEAR, cv::BORDER_CONSTANT, cv::Scalar());
for(int k = 0; k<1024*768; k++)
{
merged.data[k] = left.data[k]; //unimos los datos de las dos imagenes en una
merged.data[k + 1024*768] = right.data[k];
}
sensor_msgs::Image im, ir, il; //definimos los diferentes topicos
cv_bridge::CvImage cvi_m, cvi_r, cvi_l;
ros::Time time = ros::Time::now();
cvi_m.header.stamp = cvi_r.header.stamp = cvi_l.header.stamp =time; //transformamos desde OpenCV a ROS
cvi_m.header.frame_id = cvi_r.header.frame_id = cvi_l.header.frame_id = "camera";
cvi_m.encoding = cvi_r.encoding = cvi_l.encoding = "mono8";
cvi_m.image = merged;
cvi_r.image = right;
cvi_l.image = left;
cvi_m.toImageMsg(im);
cvi_r.toImageMsg(ir);
cvi_l.toImageMsg(il);
pub_merged.publish(im);
pub_right.publish(ir);
pub_left.publish(il);
}
int main(int argc, char **argv)
{
std::string DIR = ros::package::getPath("bumblebee2"); //hacemos un remapeo de los datos para obtener la imagen corregida
std::string PATH = DIR + "/src/remap.txt.gz";
cv::FileStorage fs;
fs.open(PATH, cv::FileStorage::READ);
fs["remaplx"] >> remaplx;
```

```
fs["remaply"] >> remaply;
fs["remaprx"] >> remaprx;
fs["remapry"] >> remapry;
fs.release();
ros::init(argc, argv, "rectifier"); //inicializamos ROS
ros::NodeHandle nh;
image_transport::ImageTransport it(nh);
image_transport::Subscriber sub = it.subscribe("/stereo_camera/image_raw", 1,
imageCallback); //publicamos los topicos y la imagen
pub_merged = it.advertise("/bumblebee2/merged", 1);
pub_right = it.advertise("/bumblebee2/right", 1);
pub_left = it.advertise("/bumblebee2/left", 1);
ros::spin();
}
```

B.4 Código para la detección de peatones en la cámara estéreo

```
#include <ros/ros.h>
#include <stdio.h>
#include <iostream>
#include "std_msgs/String.h"
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <cv_bridge/CvBridge.h>
#include <cvaux.h>
#include <math.h>
#include <cxcore.h>
#include <highgui.h>
#include <fstream>
#include <string.h>
//incluimos todos los paquetes necesarios para el funcionamiento del programa
using namespace std;
using namespace cv;
namespace enc = sensor_msgs::image_encodings;
static const char WINDOW[] = "Image window"; //habilitamos una ventana para
mostrar la imagen
class detect
{
ros::NodeHandle nh_; //definimos los datos para poder comunicarnos con otro
programa
ros::NodeHandle n;
ros::Publisher pub;
image_transport::ImageTransport it_;
image_transport::Subscriber image_sub_; //nombre del tema al que nos suscribimos
```

```
image_transport::Publisher image_pub_; //nombre del tema en el que se publica
std_msgs::String msg;
public:
detect()
: it_(nh_)
{
pub= n.advertise<std_msgs::String>("datoscamestereo", 1000) ; //le damos nombre al topico en el que escribimos
image_sub_ = it_.subscribe("/bumblebee2/right", 1, &detect::imageCb, this); //nos suscribimos al topico de la camara estereo
image_pub_= it_.advertise("/estereo/Image",1); //publicamos en el topico con el nombre seleccionado
}
~detect()
{
cv::destroyWindow(WINDOW); //al acabar, cerramos la ventana habilitada
}
void imageCb(const sensor_msgs::ImageConstPtr& msg) //llamamos al programa principal
{
char xchar[5]; //definimos algunas variables que nos seran necesarias
char ychar[5];
char anchochar[5];
char altochar[5];
sensor_msgs::CvBridge bridge; //definimos la variable para la conversion de la imagen desde ROS a OpenCV
IplImage* img = bridge.imgMsgToCv(msg,"bgr8"); //convertimos la imagen desde ROS a OpenCV usando cvbridge
Mat img2(img); //copiamos la imagen para poder trabajar con el duplicado
HOGDescriptor hog; //con estas dos sentencias declaramos el uso del histograma de gradientes orientados
hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());
vector<Rect> found, found_filtered; //aplicamos el histograma de gradientes orientados
double t = (double)getTickCount();
int can = img2.channels();
```

```
hog.detectMultiScale(img2, found, 0, Size(8,8), Size(32,32), 1.05, 2);
t = (double)getTickCount() - t;
//printf("tdetection time = %gms\n", t*1000./cv::getTickFrequency());
size_t i, j;
for (i=0; i<found.size(); i++) //aplicamos una correccion a la solucion obtenida
{
Rect r = found[i];
for (j=0; j<found.size(); j++)
if (j!=i && (r & found[j])==r)
break;
if (j==found.size())
found_filtered.push_back(r);
}
for (i=0; i<found_filtered.size(); i++)
{
Rect r = found_filtered[i]; //definimos las dimensiones del peaton
r.x += cvRound(r.width*0.1);
r.width = cvRound(r.width*0.8);
r.y += cvRound(r.height*0.07);
r.height = cvRound(r.height*0.8);
rectangle(img2, r.tl(), r.br(), cv::Scalar(0,255,0), 3); //dibujamos el rectangulo alrededor
del peaton detectado
printf("dimensiones: \n esquina superior= %i(x), %i(y) \n ancho= %i \n alto= %i \n ", r.x,
r.y, r.width, r.height);
sprintf(xchar, "%d", r.x); //transformamos los valores de posicion del rectangulo que
contiene al peaton
sprintf(ychar, "%d", r.y); //de entero a caracter para poder publicarlo
sprintf(anchochar, "%d", r.width);
sprintf(altochar, "%d", r.height);
std_msgs::String msg; //publicamos la informacion que nos interesa
std::stringstream ss;
ss << "\ndimensiones: esquina superior izquierda= " << xchar <<"(x), " << ychar <<"(y)
ancho=" << anchochar << " alto=" << altochar;
msg.data = ss.str();
```

```
pub.publish(msg);
}
imshow("Detector", img2); //mostramos la imagen con la deteccion ya realizada y en la posicion de la pantalla que nos interesa
cvMoveWindow("Detector",200,50);
cvWaitKey(2);
}
};
int main(int argc, char** argv)
{
ros::init(argc, argv, "peopledetect"); //definimos todo lo necesario para el correcto funcionamiento de ROS
detect ic;
ros::spin();
return 0;
}
```

Bibliografía

[1] DGT, www.dgt.es/Galerias/.../TEMA_11_Parte_Comun_mov_segura66g.doc [Online disponible el 17 de Abril de 2014]

[2] D. Mohan, “Social cost of road traffic crashes in India," in Proceedings First Safe Community Conference on Cost of Injury. Viborg, Denmark

[3] http://www.economiadelasalud.com/Ediciones/41/08_pdf/costes.pdf [Online disponible el 17 de Abril de 2014]

[4] DGT, http://www1.dgt.es/Galerias/prensa/2014/01/Balance-2013-Seguridad-Vial-2013.pdf [Online disponible el 17 de Abril de 2014]

[5] Comisariado europeo del automóvil http://www.cea-online.es/reportajes/seguridad.asp [Online disponible el 17 de Abril de 2014]

[6] http://www.interempresas.net/MetalMecanica/Articulos/105748-Sistemas-de-ayuda-avanzada-a-la-conduccion.html [Online disponible el 17 de Abril de 2014]

[7] http://www.coches.net/noticias/sistemas-ayuda-conductor [Online disponible el 18 de Abril de 2014]

[8] Toyota, http://www.toyotaprensa.es/prensa [Online disponible el 18 de Abril de 2014]

[9] http://googleblog.blogspot.hu/2012/08/the-self-driving-car-logs-more-miles-on.html [Online disponible el 18 de Abril de 2014]

[10] http://www.bbc.co.uk/mundo/noticias/2010/10/101010 [Online disponible el 18 de Abril de 2014]

[11] http://www.tecnocarreteras.es/web/items/1/268/ivvi-el-vehiculo-inteligente-de-la-universidad-carlos-iii-de-madrid [Online disponible el 18 de Abril de 2014]

[12] http://amigosdelaciencia.fecyt.es/?p=345 [Online disponible el 18 de Abril de 2014]

[13] http://e-archivo.uc3m.es/handle/10016/8484 [Online disponible el 19 de Abril de 2014]

[14] http://www.ecointeligencia.com/2011/11/darpa-grand-challenge-coches-sin-conductor/ [Online disponible el 19 de Abril de 2014]

[15] http://personal.cimat.mx:8181/~jbhayet/CLASES/VISION/slides/clase4.pdf [Online disponible el 20 de Abril de 2014]

[16] http://isa.umh.es/doct/pava/ModeloCamara.pdf [Online disponible el 20 de Abril de 2014]

[17] http://www.sitopcar.es/modulos/descargas/manuales/Introduccion_Fotogrametria_Digital.pdf [Online disponible el 20 de Abril de 2014]

[18] http://www.cesfelipesegundo.com/revista/articulos2011/Guerrero,%20J.M.pdf [Online disponible el 20 de Abril de 2014]

[19] http://web.eecs.umich.edu/~silvio/teaching/EECS598_2010/slides/09_28_Grace.pdf [Online disponible el 21 de Mayo de 2014]

[20] http://eprints.ucm.es/16073/1/Interaccion_persona-computador_basada_en_el_reconocimiento_visual_de_manos.pdf [Online disponible el 10 de Mayo de 2014]

[21] http://www.grc.upv.es/docencia/tdm/practicas/P1.pdf [Online disponible el 21 de Mayo de 2014]

[22] http://zaguan.unizar.es/TAZ/EINA/2012/8012/TAZ-PFC-2012-395.pdf [Online disponible el 21 de Mayo de 2014]

[23] W. Garage, ‘’OpenCV Wiki” https://www.willowgarage.com/pages/software/opencv [Online disponible el 15 de Abril de 2014]

[24] Manejo de las bibliotecas OpenCV http://www.electron.frba.utn.edu.ar/~afurfaro/Info1/Opencv/opencv.pdf [Online disponible el 15 de Abril de 2014]

[25] http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf [Online disponible el 21 de Mayo de 2014]

[26] http://es.wikipedia.org/wiki/Middleware [Online disponible el 21 de Mayo de 2014]

[27] http://e-archivo.uc3m.es/bitstream/handle/10016/10332/PFC_Victor_Martinez_Leon_FINAL.pdf?sequence=1 [Online disponible el 21 de Mayo de 2014]

[28] http://www.slideshare.net/dobby74/sistemas-operativos-8219281 [Online disponible el 21 de Mayo de 2014]

[29] http://wiki.ros.org/ROS/Tutorials [Online disponible el 21 de Mayo de 2014]

[30] http://wiki.ros.org/microstrain_3dmgx2_imu/Tutorials/UsingTheIMUNode [Online disponible el 21 de Mayo de 2014]

[31] http://wiki.ros.org/camera1394 [Online disponible el 21 de Mayo de 2014]

[32] http://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf [Online disponible el 24 de Mayo de 2014]

[33] http://www.puertos.es/ayudas_navegacion/sistemas_navegacion_por_satelite/Sistemas_GPS_y_DGPS.html [Online disponible el 21 de Mayo de 2014]

[34] http://www.costanerauno.com.ar/preguntasfrecuentes/Qu%C3%A9+es+y+qu%C3%A9+significa+NMEA_15_86.htm [Online disponible el 21 de Mayo de 2014]

[35] http://wiki.ros.org/cereal_port [Online disponible el 21 de Mayo de 2014]

[36] V. Jiménez Monje, ‘’Detección y localización de obstáculos en entornos urbanos mediante visión estéreo", 2011. [PFC Universidad Carlos III de Madrid]

[37] Gary Bradski, Adrian Kaehler, “Learning OpenCV: Computer Vision with the OpenCV Library” [O´Reilly]